Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Irányítástechnika és Informatika Tanszék

Adatgyűjtő robotok pályatervezése érzékelő csomópontok között

TDK dolgozat

Készítette:

Olasz-Szabó Sára

Konzulens:

dr. Harmati István

2021

Tartalomjegyzék

Kivonat

Manapság egyre gyakoribb, hogy egy adott területről folyamatos adatgyűjtésre van szükség. Például egy őrzött területen behatoló mozgásának feltérképezése, egy erdőben a vadak követése, vagy egy mezőgazdasági területen az időjárási elemek megfigyelése. Az adatgyűjtés vezeték nélküli szenzoros hálózatokkal (WSN) tehető meg a legegyszerűbben. Azonban a hálózaton belüli adatküldés nagy energia felhasználással jár, ami a hálózat élettartalmát jelentősen csökkentheti. Ezt a problémát úgy próbáljuk meg orvosolni, hogy az adott érzékelő csomópontok közt az adatátvitelt robotokkal valósítjuk meg, ilyen módon nem kell távolra adatot küldeni, ami jelentősen csökkenti az energiafelhasználást, és ennek következtében jelentősen növeli a hálózat élettartalmát.

Az adatgyűjtést egy olyan terepen valósítjuk meg, ahol az érzékelő csomópontok szétszórtan helyezkednek el, és közöttük különböző terepi tárgyak, akadályok fordulhatnak elő, amelyet a robotoknak ki kell kerülni. Az útvonal tervezés során a cél az, hogy a bázis-csomópontból kiindulva az összes érzékelő-csomópontot bejárja a robot, azokról az adatokat töltse le, majd a bázishoz visszaérve azokat töltse fel további feldolgozásra. Tehát a megtervezett útvonalon a robot ismételten végig halad a folyamatos adatgyűjtés érdekében. Természetesen fontos, hogy a megtervezett út hossza, és ennek következtében a bejárás ideje minimális legyen annak érdekében, hogy a begyűjtött adatok késleltetési ideje a lehető legkevesebb legyen.

Az adatgyűjtés hatékonyságának növelése érdekében a vizsgálandó terepen egyszerre több együttműködő robot útvonalát is megtervezem. Ebben az esetben a minimális útvonal hossz mellett az is fontos, hogy a robotok megközelítőleg azonos időtartam alatt járják be a saját útvonalukat, hiszen így az egyes érzékelő csomópontok adatait megközelítőleg azonos időközönként vizsgálhatjuk meg.

Az útvonal tervezés során nagy hangsúlyt fektetek az akadályok kezelésére. Amennyiben a terepen sok vagy nagyméretű akadályok fordulnak elő, azok elkerülése hosszas lehet, így ez egy kritikus pontja a minimális útvonal megtalálásának.

Abstract

Nowadays there is a more and more common need for continuous data collection on a specified area. For example, this can be the tracing of the movements of an intruder on a protected area, the following of games in a forest, or the observation of weather on an agricultural area. The simplest way for such data collection is using wireless sensor networks (WSN). However, data transmission within the network uses great amount of energy that may reduce significantly the lifetime of the network. We try to solve this problem by applying robots for data transmission between sensor nodes. This way there is no need to send data to remote destinations so the amount of energy used decreases significantly which increases the lifetime of the network.

Data collection is happening on an area scattered with sensor nodes. There can be different objects/obstacles between sensor nodes and the robots should avoid them. The goal of path planning is that the robot visits all nodes starting from the base node, downloads the data from the nodes and uploads the collected data for further data procession after returning to the base node. So the robot will use the planned path repeatedly in order to collect data continuously. Naturally, it is important that the length of the planned path and this way the total time spent with data collection be as short as possible in order to minimize the delay time of the collected data.

In order to increase the efficiency of data collection I plan the path of more cooperating robots on the area in question. In this case, beside minimizing the length of the path it is also important that the robots spend approximately the same amount of time on their route since this way one can examine all sensor node data by almost on the same interval of time.

During path planning I greatly emphasize the handling of obstacles. If the area contains many or large obstacles, the robots may spend many time for avoid them so this is a critical point of finding the minimal path.

1. fejezet

Bevezetés

Napjainkban egyre gyakoribb, hogy egy nagyobb területről folyamatos adat gyűjtésre van szükség. Ezt legtöbbször vezeték nélküli szenzoros hálózatokkal (WSN) valósítják meg [1] [2]. A távoli adatküldés nagy energia disszipációval jár, amely csökkenti a hálózat élettartalmát. Ezért az adatátvitelt adatgyűjtő robotok segítségével hajtjuk végre [3] [4]. Azonban mivel a robotoknak van maximális haladási sebességük, az adat késleltetési ideje jelentősen megnő. Az adatgyűjtő robotok segítségével az adatküldés kis távolságokra is elérhetővé válik, amely több előnyt is magában rejt. Csökkenti az adatátvitel energia igényét és egyúttal az átvitel hibaszázalékát és növeli a hálózat megbízhatóságát, amely több alkalmazási területen is fontos szempont [5].

A robotoknak be kell járniuk az összes érzékelő csomópontot, majd a bázisállomásra visszatérve fel kell tölteniük a begyűjtött adatot. A robotok útvonalának a megtervezése egy fontos pont, hiszen az útvonal hossza közvetlenül befolyásolja az adatok késleltetési idejét. A terepen elhelyezkedő különböző akadályok tovább nehezítik az útvonal tervezést. Annak érdekében, hogy az adatgyűjtés biztosan sikeresen menjen végbe szükség van arra, hogy meghatározzuk a megfelelő út kritériumait. Az út legyen kellően sima, ütközésmentes mind az érzékelő csomóponttokkal mind az akadályokkal, zárt és elég időt biztosít az adatok letöltésére [6]. Az út simaságára a robot kinematikai és dinamikai tulajdonságai miatt van szükség. Az akadályok és az érzékelő csomópontok körül egy biztonsági sávot létesítünk az eszközök épségének érdekében. A robot folyamatos adatgyűjtést hajt végre, ezért szükséges az út zártsága. Az adatgyűjtést egy egy kerekű robot, Dubins-jármű [7] teszi meg, amely csak konstans haladási sebességgel és korlátozott szögsebességgel tud haladni. Az adatgyűjtési idő csökkentése érdekében célszerű egyszerre több együttműködő robotnak útvonalat készíteni. A feladat megoldását és megfogalmazását a [6] tartalmazza, ezt használom a dolgozatom kiindulópontjaként.

A munkám során először elkészítettem az útvonal tervezést a [6] irodalomban leírt öt lépést tartalmazó SVPP–Shortest Viable Path Planning algoritmus szerint egy adatgyűjtő robotot alkalmazva. Ennek egyes lépéseinek megvalósításához algoritmusokat készítettem el, ilyen a SVPP algoritmus negyedik lépése helyett alkalmazandó fagráf létrehozására kidolgozott. A fagráfban a legrövidebb útvonal egy egyszerű gráf-bejárási algoritmussal [8] [9] megoldható lesz.

Az útvonal tervezés során nagy hangsúlyt fektetek az akadályok kezelésére. Amennyiben a terepen nagyszámú vagy nagyméretű akadályok fordulnak elő, azok elkerülése hosszas lehet, így ez egy kritikus pontja a minimális útvonal megtalálásának. Ennek a jelentősége tovább nő több robot egyidejű alkalmazása esetén. Ezen oknál fogva, az akadályok minél hatékonyabb kikerülése érdekében új algoritmusokat készítettem.

A feladat megoldását több robot egyidejű alkalmazásával is megvalósítottam. Ilyenkor a cél szintén a bejárási idő csökkentése, azonban a minél hatékonyabb útvonal tervezés

miatt fontos, hogy a robotok közel azonos idő alatt járják be a rájuk bízott érzékelő csomópontokat. Erre három algoritmust implementáltam. A k-SPLITOUR [10] és a k-SVPP a [6] irodalomban kerültek bemutatásra. Az SVPP algoritmus helyett ebben a részben a módosításaimmal ellátott M-SVPP (Modified-SVPP) algoritmust használom. A harmadik, több robotot igénybe vevő megoldást a k-means algoritmus [11] ötletét alkalmazva fejlesztettem ki.

A dolgozatomban először összefoglalom a problémát amit meg szeretnénk oldani, kezdve a Dubins-jármű mozgásának leírásával, majd a feladat pontos megfogalmazásával, feltételezésekkel folytatom 2.3 fejezet. Ezt követően egy robotra felvázolom az úttervezés lépéseit a 3 fejezetben. Először összefoglalom [6] megoldását, majd bemutatom a saját megfontolásaimat, algoritmusaimat. A 4 fejezetben ismertetem a három megoldást a több robotra való útvonaltervezésre. Majd az utolsó, 5 fejezetben bemutatom a szimulációs eredményeket, először egy robotot alkalmazó esetre, ahol felvázolom az útvonal tervezés lépéseit illetve összehasonlítom az munkám eredményét [6] irodalom algoritmusaival kapottéval. Majd a több robotot alkalmazó algoritmusokat hasonlítom össze.

2. fejezet

A probléma leírása

A probléma leírása során először kitérek az útvonalat bejáró Dubins jármű modellezésére, majd ismertetem az alap feltételezéseket, amelyekkel a megoldás során élni fogok.

2.1. A Dubins jármű

A Dubins jármű [7] egy egy kerekű robot, amely állapotát le lehet írni az $X = (x, y, \theta)$ hármassal, ahol az $x, y \in \mathbb{R}^2$ a robot pozícióját, θ a robot irányát jelöli. A jármű konstans v sebességgel halad, és mozgását az u szögsebességgel lehet irányítani. A robot dinamikai leírása

$$\dot{x} = v\,cos\,\theta\,(t) \tag{2.1}$$

$$\dot{y} = v\,sin\,\theta\,(t) \tag{2.2}$$

$$\dot{\theta} = u(t) \in [-u_M, u_M] \tag{2.3}$$

ahol u_M a maximális szögsebességet jelöli, tehát

$$|u(t)| \leqslant u_M. \tag{2.4}$$

Mivel a robot csak konstans sebességgel és korlátozott szögsebességgel tud haladni, irány változtatás során mindig egy R sugarú körív mentén mozog. Ennek minimális sugara

$$R_{min} = \frac{v}{u_M}, \tag{2.5}$$

tehát a jármű ennél kisebb sugárban nem tud fordulni.

Figyelembe véve az egykerekű robot kinematikai tulajdonságait három különböző mozgás primitívet [12] határozunk meg az optimális útvonal készítéséhez. Mivel a v haladási sebesség konstans, ezeket a primitíveket a szögsebesség alapján különböztethetjük meg.

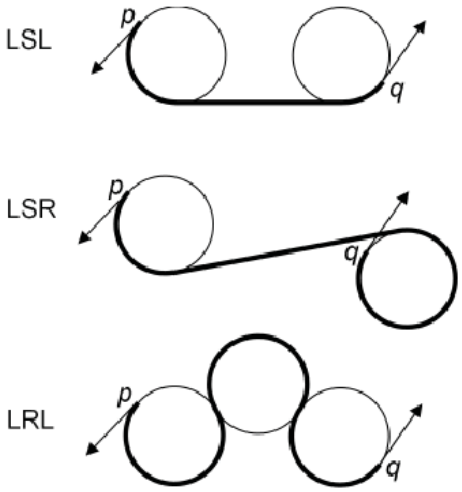
Jelölés	u
S	0
L	+1
R	-1

2.1. táblázat. A Dubins Jármű mozgás primitívei az optimális útvonal készítéshez.

A 2.1 táblázatban szereplő S, L, és R sorra az egyenesen haladást, balra és jobbra kanyarodást jelölik az R_{min} fordulási sugárral. Az optimális Dubins-útvonal tervezésekor a

robot vagy nem kanyarodik, vagyis a szögsebessége zérus, vagy abszolútértékben maximális szögsebességgel fordul valamely irányban, tehát $u \in \{-1, 0, 1\}$.

A Dubins-görbék primitív hármasokból álló szavak, amelyekkel leírható az optimális útvonal. Ezek: $\{LRL, RLR, LSL, RSR, LSR, RSL\}$, amelyeket a 2.1 ábra szemléltet.



2.1. ábra. A legrövidebb útvonalat leíró Dubins-görbék közül három szemléltetése, a többi tükrözéssel ebből előállítható. [13]

A pályán elhelyezkedő akadályok a feladatmegoldása során az optimális útvonalat blokkolhatják, ezért a Dubins-görbékkel való útvonal tervezés ebben az esetben nem, vagy csak nehezen megvalósítható. Ennek következtében a feladatot más, azonban hasonló módosított módszerrel oldom meg.

2.2. A feladat leírása

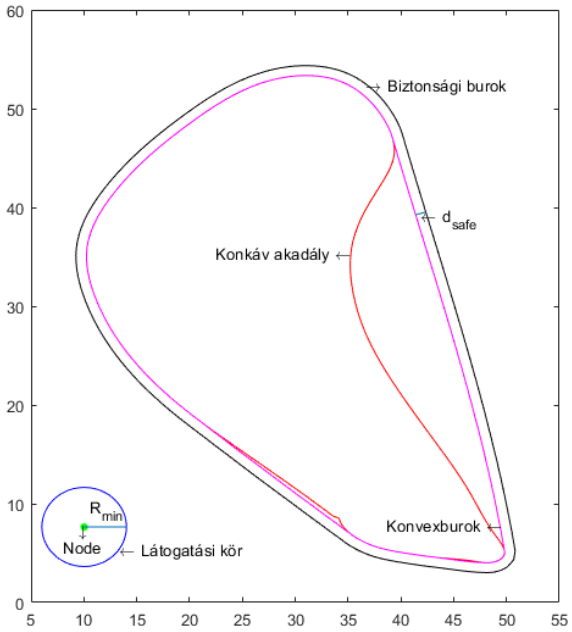
A bejárandó terepen található n darab adatgyűjtő csomópont amelyekből egy bázis állomás (s_1) és $n-1$ érzékelő csomópont ($s_i, i \in [2, n]$). Az érzékelő csomópontoknál az adat letöltésre kerül, a bázis állomásnál feltöltésre. Mivel a robot a bázis állomástól kapja meg a bejárandó útvonalat, ezért szükségszerűen onnan kell kiindulnia. A terep az érzékelő csomópontokon kívül tartalmaz még m darab elkülönülő akadályt ($o_j, j \in [1, m]$), amelyeket a robotnak el kell kerülnie az útvonal bejárása során. Azonban azt az útvonal tervezés közben feltételezzük, hogy pontosan ismerjük mind az akadályok, mind a csomópontok helyét, és előbbieknél az alakjukat, utóbbiaknál az adatmennyiséget, amelyet majd a robotnak le kell töltenie, illetve a bázis állomásnál feltöltenie.

2.3. A probléma megoldása során tett feltételezések

A fent leírtak szerint a bejárandó terepen az akadályok helyét és alakját pontosan ismerjük a feladat megoldása során. Továbbá minden akadálynak simának kell lennie, és a görbületének minden pontban olyannak, hogy a robot a minimális R_{min} fordulási sugarával azt követni tudja. A robot a mozgása során, ha az akadály kerületén kényszerül menni az akadály kikerülése érdekében, akkor annak felszínétől egy előre meghatározott d_{safe} biztonsági távolságra halad. Ugyanakkor, ha az akadály egy konkáv alakzat, akkor a felszínén a robot úgy tud a legrövidebb úton menni, ha az akadálynak a konvex burkán mozog.

Ahhoz, hogy egy utat járhatónak nevezhessünk, a következő öt kritériumnak kell teljesülnie: 1.) sima, és ütközésmentes 2.) a bázis és az érzékelő csomópontokhoz, illetve 3.) az akadályokhoz, 4.) zárt és 5.) biztosít elég időt az egyes érzékelő csomópontoknál az adatok le- és feltöltéséhez. A simaság azért szükséges, mert a robot rendelkezik egy minimális fordulási sugárral, illetve csak konstans sebességgel tud haladni, így a hirtelen változásokat nem tudná követni a mozgása során. Mivel a robot sem az akadályokon sem a csomópontokon nem tud átkelni, azokhoz érve sérülést szenved, szükséges az, hogy a tervezett út ütközés mentes legyen. A robotnak az adatgyűjtést periodikusan és folyamatosan kell végrehajtania, ezért ésszerű az a döntés, hogy a tervezett út zárt. Az utolsó feltétel pedig azt biztosítja, hogy a robot az összes adatot le és fel tudja tölteni az egyes csomópontoknál.

Az egyes csomópontoknál a látogatási kör (visiting circle), egy olyan kör, amelynek középpontja az adott csomópont, sugara pedig az R_{min} távolság. Ha az R_{min} távolságnál kisebb sugarú kört vennénk, akkor a robot nem tudná megkerülni az adott csomópontot, hiszen nem tud ennél az értéknél kisebb sugárban fordulni. Ha a robotnak sok időbe telik, mire letölti az összes adatot, akkor ezen a körön többször végig haladva tudja megtenni. Az R_{min} távolság felfogható úgyis, mint a csomópontok biztonsági margója, mint az akadályoknál a d_{safe}. A látogatási köröket $C_i, i \in [1, n]$, míg az akadályok konvexszé alakított és biztonsági sávval ellátott alakját $\delta o_j, j \in [1, m]$ jelöli. Az elnevezéseket 2.2 ábra magyarázza.

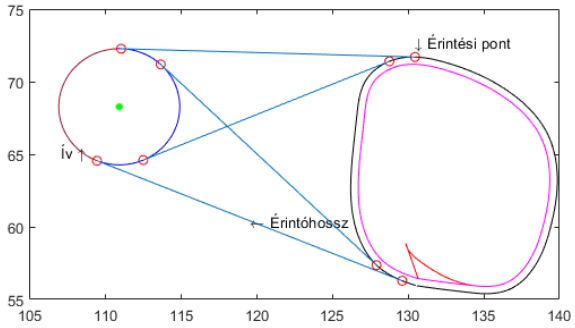


2.2. ábra. A csomópontok látogatási köre és az akadály konvex- és biztonsági burka

A következő feltétel az, hogy az egyes terepen levő elemek, csomópontok és akadályok egymással nem érintkezhetnek, az akadályok és a csomópontok közt is minimálisan annyi helynek biztosítottnak kell lennie, hogy a robot ütközés nélkül el tudjon haladni köztük, tehát két akadály közt $2\,d_{safe}$, két csomópont közt $2\,R_{min}$ és egy akadály és egy csomópont közt $d_{safe} + R_{min}$ távolságot kell legalább biztosítani.

Ha az adott terep elrendezés megfelel a fenti követelményeknek, akkor elkészítendő az úgynevezett érintőgráf. Ez egyrészt olyan szakaszokat tartalmaz, amely egyszerre két objektumnak (csomópont vagy akadály) is az érintője, ugyanakkor nem metsz más objektumokat. Az egyes objektumok és a hozzájuk tartozó érintő közös pontját érintési pontnak nevezzük, míg az ugyanazon az objektumon lévő érintési pontok közti görbét ívnek. A két objektum közös érintő által létrehozott érintési pontok távolságát fogom érintőhossznak nevezni a továbbiakban. Az elnevezéseket a 2.3 ábra mutatja. Az érintő gráf létrehozásához, az összes olyan érintő meghatározandó, amely egyszerre tartozik két objektumhoz, azonban más objektumokat nem metsz. Tehát $G(V, E)$ jelöli az érintőgráfot (Tangent Graph), ahol V tartalmazza az érintési pontokat és E az érintőket illetve az íveket, együttesen az éleket.

A [6] irodalom ennél a lépésnél az érintőgráfba azokat az érintőket sem veszi be, amelyek az egyes látogatási köröket metszik, ezt a továbbiakban **L1** feltétel fogom nevezni. A munkám során azonban, ezeket az éleket abban az esetben beveszem az $G(V, E)$ érintőgráfba, ha az érzékelő csomóponttól minimum d_{safe} távolságra haladnak el, **L2** megkötés.



2.3. ábra. Az érintési pont, az érintő és az ívhossz

Ha a robot az érintőgráf elemein halad végig az útja során, akkor az ütközés mentesség (2.) és 3.) feltétel) automatikusan teljesül, azonban az út simasága még nem biztosított. Erre ad megoldást a következő megkötés, az irányítottsági kényszer. Ez azt mondja ki, hogy a robot irányának az egyik élre való belépéskor ugyanolyannak kell lennie, mint amilyen irányban elhagyta az utolsó élt. Tehát ha $edge_1, edge_2 \in E$ és a közös érintő pont p, akkor jelölje $\theta^p_{edge_1}$ és $\theta^p_{edge_2}$ a robot irányítottságát rendre az $edge_1$ él végén, és az $edge_2$ él elején a p pontban. Ekkor az irányítottsági kényszer:

$$\theta^p_{edge_1} = \theta^p_{edge_2}. \tag{2.6}$$

Ilyen módon az út simasága is biztosított.

Ahhoz, hogy a megfelelő idejű és minőségű adatletöltést tudjuk biztosítani, az egyes csomópontokról csak akkor veszünk fel információt, ha a robot az éppen soron következő csomópont látogatási körén mozog. A robot akkor is tudna adatot letölteni, ha a látogatási körön kívül mozog, azonban ekkor csökkenne az adatátvitel gyorsasága és minősége is. Ráadásul a közelebbi adatcsere kisebb energia felhasználással jár. A csomóponton felhalmozott adat mennyisége és az adatátviteli sebesség együttesen határozza meg a szükséges adatletöltési időt. Az adatátviteli sebesség függ a csomópontoktól való távolságtól, annak

egy nemnövekvő függvénye $r(d)$ (bit/s):

$$r(d) = \begin{cases} r_1,\ 0 < d \leqslant d_1 \\ r_2,\ d_1 < d \leqslant d_2 \\ \vdots \\ r_i,\ d_{i-1} < d \leqslant d_i \\ \vdots \\ 0,\ d > d_{max} \end{cases} \tag{2.7}$$

ahol d_{max} az a maximális távolság, ahonnan még a csomópontról adatot lehet letölteni. Jelölje g_i az s_i csomóponton felhalmozott adatmennyiségét. Ekkor a szükséges kontakt idő δ_i a következő módon számítható:

$$\delta_i = \frac{g_i}{r(R_{min})}. \tag{2.8}$$

Ha a robot v sebességgel mozog, akkor a minimális ívhossz l_i ami az adatletöltéshez szükséges:

$$l_i = \delta_i v = \frac{g_i v}{r(R_{min})}. \tag{2.9}$$

Mivel a látogatási körhöz csak érintési pontoknál lehet bejutni és csak azokon lehet kijutni, illetve mivel a robot csak konstans v sebességgel mozog az irányítottsági kényszer betartásával, ezért általában nem biztosítható az, hogy a robot pontosan annyi ideig haladjon az adott csomóponthoz tartozó látogatási körön amennyi az adatletöltéshez szükséges. Ez oknál fogva előfordulhat, hogy a robotnak a csomópont körül több kört is meg kell tennie. Ennek a meghatározása bevezetésre kerül a és b paraméter, amelyek a be- és kimeneti érintési pontok közt megteendő út hosszának leírásához használatosak.

$$a = \left\lfloor \frac{g_i v}{r(R_{min}) 2\pi R_{min}} \right\rfloor, \tag{2.10}$$

$$b = \frac{g_i v}{R_{min}} - a \cdot 2\pi R_{min}, \tag{2.11}$$

ahol $\lfloor \cdot \rfloor$ az egészrész függvényt jelöli. Ha adva van a látogatási kör $C_i, i \in [1, n]$ és az összes ív C_i mentén, akkor egy adott p_1 érkezési és p_2 elhagyási pont közt a megteendő út C_i mentén a következő módon számítható:

$$|\widehat{p_1 p_2}|_s = \begin{cases} a \cdot 2\pi R_{min} + |\widehat{p_1 p_2}|, & \text{ha } b \leqslant |\widehat{p_1 p_2}| \\ (a+1) \cdot 2\pi R_{min} + |\widehat{p_1 p_2}|, & \text{ha } b > |\widehat{p_1 p_2}| \end{cases}. \tag{2.12}$$

Tehát $|\widehat{p_1 p_2}|_s$ az az ívhossz amelyet a robot ténylegesen megtesz. Az adott C_i látogatási körhöz p_1 ponton érkezik, megtesz a vagy $a+1$ kört, majd a p_2 ponton távozik a körből. A $G(V, E)$ gráf a továbbiakban a módosított érintő gráfot jelöli, ahol V továbbra is az érintési pontokat tartalmazza, azonban E az ívhosszakat a fent kiszámolt (2.12) módon tárolja.

A bejárandó út zártságát, illetve az összes csomóponton való áthaladást a DTSPN (Dubbins Travelling Salesman Problem with Neighbourhoods) [14] segítségével kerül megoldásra. Adva van $C = \{C_1, \cdots, C_n\}$ látogatási körök és ezeknek egy $\Sigma = \{\sigma_1, \cdots, \sigma_n\}$ permutációját keressük. Definiálunk egy leképezést $\mathcal{P} : SE(2) \longrightarrow \mathbb{R}^2$, azaz $\mathcal{P}(X) = (x, y)$. Tehát keressük az optimális megoldást az összes lehetséges Σ és X konfigurációból, amely-

re:

$$\min_{\Sigma,X} |\mathcal{L}(X_{\sigma_n}, X_{\sigma_1})| + \sum_{i=1}^{n-1} |\mathcal{L}(X_{\sigma_i}, X_{\sigma_{i+1}})|$$

$$\mathcal{P}(X_{\sigma_i}), \quad i \in [1, n] \tag{2.13}$$

ahol $\mathcal{L}(X_1, X_2)$ egy járható minimális hosszal rendelkező útrész X_1 és X_2 közt, ahol $\mathcal{P}(X_1) \in C_{i_1}, \mathcal{P}(X_2) \in C_{i_2}, i_1, i_2 \in [1, n]$ és $i_1 \neq i_2$, $|\mathcal{L}(X_1, X_2)|$ megadja ezt a legrövidebb úthosszat. A járható útrész egy konzisztens részhalmaza E-nek, amelyben bármely két egymásutáni él a járható útrészleten rendelkezik közös érintési ponttal, ahol az irányítottsági kényszer teljesül. A $\mathcal{L}$ függvény a fenti 1.), 2.), 3.), és 5.) feltételnek megfelelően működik és, ha ehhez (2.13), annak megoldása során kielégítésre kerül a 4.) feltétel is, akkor a legrövidebb járható út meghatározható az egykerekű adatgyűjtő robotra.

3. fejezet

A legrövidebb járható út tervezése

A DTSPN [14], mint maga a TSP (Travelling Salesman Problem- utazó ügynök probléma), NP-hard. Ráadásul a DTSPN végtelen számú megoldási lehetőséggel rendelkezik tehát egzakt megoldására nincs lehetőség. Azonban az előző fejezetben taglalt (2.13) probléma véges számú lehetőséget ad, felfogható úgy, mint mintavételezett DTSPN, azzal a kiegészítéssel, hogy a terepi objektumok nem lapolódhatnak át. Az így keletkező probléma továbbra is NP-hard, így a megoldása úgy, hogy az összes lehetőséget kiszámoljuk, és abból a legkedvezőbbet választjuk, nagyon számítás, és így időigényes.

A fent említett okok miatt a problémát érdemes algoritmikusan megoldani, amelyet a [6] irodalomban SVPP (Shortest Viable Path Planning– legrövidebb járható út tervezés) algoritmusnak neveznek. Ennek során először létrehozandó egy Σ permutáció a csomópontok közt $G(V, E)$ érintőgráf alapján, majd ezt követően egy $G'(V', E')$ egyszerűsített érintő gráf kerül elkészítésre. Végül pedig a meghatározott Σ permutáció és az $G'(V', E')$ egyszerűsített érintőgráf felhasználásával készítendő egy irányított fagráf, amelyben a legrövidebb utat megkeresve rendelkezésre fog állni a probléma megoldása. Fontos kiemelni, hogy az algoritmus nem feltétlenül az optimális megoldást adja, de annak egy jó közelítése.

A fejezetben először bemutatom a SVPP algoritmus 3.1 alfejezetben, majd részletesen kifejtem általam tett módosításokat, újításokat 3.2 alfejezetben. Az itt leírtakkal megvalósított algoritmust, Modified-SVPP (M-SVPP) algoritmusnak fogom nevezni.

3.1. A SVPP algoritmus összefoglalása

Az SVPP– Shortest Viable Path Planning algoritmus a [6] irodalomban található. Az algoritmus rövidített vázlatát az **Algoritmus-1**–SVPP mutatja. Az első lépésben meghatározza a bázis és az érzékelő csomópontok Σ sorrendjét. Ehhez egy olyan irányított gráfot konstruál, amelynek a csúcsai a csomópontok, éleit pedig a következőképpen számítja. Először is meghatározza a két csomóponthoz tartozó látogatási körökön fekvő négy érintőt és ezeknek veszi az átlag hosszát, majd ehhez hozzáadja a második csomóponthoz tartozó adatletöltéshez szükséges ívhosszat. Ilyen módon egy aszimmetrikus irányított gráf áll majd rendelkezésre. Ebből az irányított gráfból egy ATSP megoldó [15] segítségével meghatározza a csomópontok bejárási sorrendjét. Ebben a lépésben az akadályoktól eltekintünk, tehát azok az érintők amelyek akadályokon haladnak át használhatóak, nem kerülnek eltávolításra a $G(V, E)$ gráfból.

A meghatározott Σ permutáció és az érintő gráf alapján elkészíthető az egyszerűsített érintő gráf. Először is az egyszerűsített érintő gráfba bekerülnek azok az érintők, melyek kettő olyan látogatási kör között futnak amelyek a permutációban egymást követik és amelyek közt valóban létezik él, vagyis nem blokkolja ezeket akadály. Természetes bekerülnek az ezekhez tartozó ívek is. Ha egy akadály blokkol egy érintőt két olyan csomópont

Algoritmus 1–SVPP

1. A csomópontok Σ sorrendjének meghatározása az akadályok figyelembe vétele nélkül –ATSP (aszimmetrikus utazó ügynök probléma)– a $G(V, E)$ érintőgráf alapján
2. A blokkoló akadályok hozzáadása a csomópontok sorrendjéhez (Σ')
3. Egyszerűsített érintőgráf létrehozása úgy, hogy azokat az éleket és csúcsokat megtartjuk amelyeket megkövetel a Σ' permutáció, a maradékot pedig töröljük $\longrightarrow G'(V', E')$
4. Az $G'(V', E')$ egyszerűsített érintőgráf alapján T fa-gráf létrehozása
5. P legrövidebb útvonal keresése mind óramutató járásával megegyező, mind ellentétes irányban

között, amelyek a permutációban egymást követik, akkor azokat az éleket is beveszi az egyszerűsített érintő gráfba, amelyek az akadály burkolójára illeszkednek, az akadályt pedig beveszi a permutációba a megfelelő helyre, az adott két csomópont közé. Ezt addig ismétli amíg nem talál több blokkoló akadályt két egymást követő érzékelő csomópont között. Ha egy akadály több csomópontpárt is blokkol, akkor az akadályt több helyre is beleveszi a permutációba. Tehát az akadályok burkolójával is kiegészített permutáció Σ' és az egyszerűsített érintő gráf $G'(V', E')$, ahol igaz, hogy $V' \subseteq V$, $E' \subseteq E$ és $\Sigma \subseteq \Sigma'$. n' jelölje a Σ' permutáció elemeinek a számát, ekkor a (2.13) a követező módon írható fel:

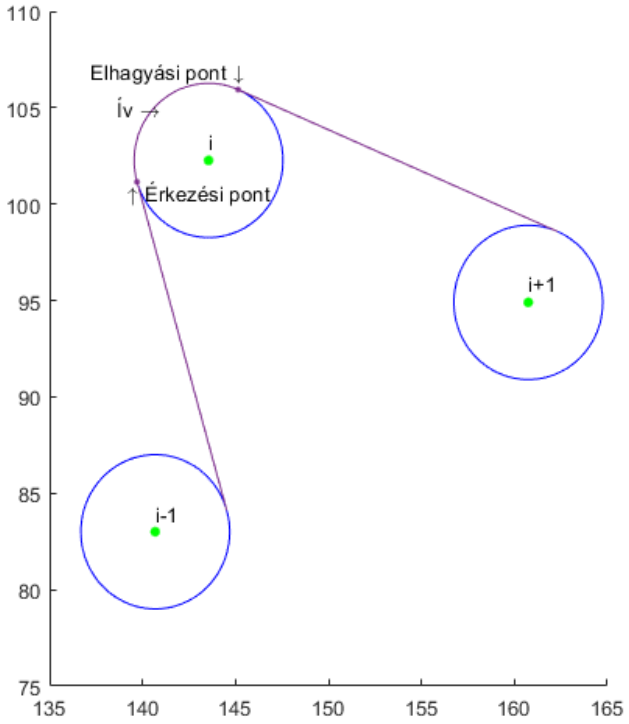
$$\min_X |\mathcal{L}'(X_{\sigma'_{n'}}, X_{\sigma'_1})| + \sum_{i=1}^{n'-1} |\mathcal{L}'(X_{\sigma'_i}, X_{\sigma'_{i+1}})|$$

$$\mathcal{P}(X_{\sigma'_i}), \quad i \in [1, n'], \tag{3.1}$$

ahol $\mathcal{L}' = \mathcal{L}$. Tehát az **Algoritmus-1** leírásban feltüntetett 2-3 lépést egymással párhuzamosan hajtható végre.

A Σ' permutációkban egymás után szereplő objektumok közti távolság mindig két összetevőből áll. Egyrészt az első objektumon az ívhosszból, amíg a meghatározott következő érintési pontig eljut a robot az előző érintési pontból indulva az objektum kerületén, –csomópontoknál a megfelelő számú kör megtétele után– másrészt pedig az adott két objektumon fekvő érintési pont közötti érintőhossz. Tehát ahhoz, hogy meg tudjuk határozni az i-edik és az $i+1$- edik ($i, \in [2, n'-1]$) objektum közötti távolságot tudnunk kell, hogy az $i-1$ -edik és i-edik objektum között futó érintő az i objektum kerületén melyik érintési pontjába fut be, hiszen erre is szükség van az i-edik objektum kerületén megtett ívhossz kiszámításához (3.1 ábra).

Két objektum közötti utat, úgy ábrázoljuk az irányított fagráfban, hogy két részre szedjük, az objektum ívén való, és a két objektum közötti érintőn való haladásra, az iránya pedig mindig a következő bejárandó útszakasz felé mutat. A gráf csúcspontjai továbbra is az érintési pontok lesznek, és köztük lévő ívhossz vagy érintőhossz lesz az él. Az élek berajzolásakor ügyelni kell az irányítottsági feltétel teljesülésére (2.6). Minden objektumból négy él indul ki a következő objektum felé –indulási konfiguráció– és ugyanannyi érkezik be az előzőből –érkezési konfiguráció–, ebből kifolyólag minden objektum nyolc csúcspontba képződik le. A bázis állomás a kiinduló csomópont, ez a fagráf első eleme, innen indul a bejárás, méghozzá úgy, hogy a bázis állomás látogatási körén jutunk el az első érintési pontokhoz, a kiinduló iránytól függően. A zárt út biztosítása érdekében a bázis állomást a fagráf utolsó elemének is választjuk. A fagráf mind a pozitív mind a negatív indulási iránynak megfelelően elkészítésre kerül. Egy érkezési konfigurációból indulva két választási lehetőség van a következő útszakasz kiválasztására az irányítottsági kényszer miatt. Tehát a kezdőpontból indulva a permutáción végighaladva $2^{n'}$ választási lehetőség lenne, azonban



3.1. ábra. A permutációban egymás után szerepló objektumok közti távolság két összetevője: ív- és érintőhossz

a kezdő és a befejező iránynak azonosnak kell lennie a folyamatos körüljárás miatt, tehát az utak száma a kezdőpontból a végpontig $2^{n'-1}$.

A fagráfban való legrövidebb útvonal keresést dinamikus programozással oldja meg a [6] irodalom.

3.2. Módosítások, implementálás

A munkám során alapvetően követem az előbb összefoglalt SVPP algoritmust [6], azonban több módosítással élek amelyek egyrésze implementálásbeli, másrésze az útkeresés hatékonyságát növelheti. Az itt bemutatott módosításokkal megvalósított SVPP algoritmust Modified-SVPP algoritmusnak nevezem.

3.2.1. Az érintőgráf létrehozása

Az érintő gráfba a [6] azokat az érintőket sem veszi be, amelyek az egyes látogatási köröket metszik, azonban az érzékelő csomópontok mellett a robot elhaladhatna minimum d_{safe} távolságra. Ez oknál fogva, a megvalósításom során azokat az érintőket amelyek ugyan metszenek látogatási kört de az érzékelő csomóponttól minimum d_{safe} távolságra haladnak el, szintén beveszem az érintőgráfba (**L2**). Így az egyes esetekben a robot rövidebb útvonalon fog tudni haladni.

3.2.2. A csomópontok bejárási sorrendjének meghatározása

A csomópontok bejárási sorrendjének meghatározása során csak a csomópontokat vesszük figyelembe, az akadályoktól ebben a részben eltekintünk. Tehát adva van az összes csomópont $s_i \in [1, n]$, és az összes olyan érintő két csomópont látogatási köre közt, amely nem halad át csomópontok köré húzott d_{safe} sugarú körön **L2**. Az [6] itt a látogatási körökkel való metszést figyeli **L1**. Mivel az akadályoktól eltekintettünk, azokon áthaladhatnak az érintők ebben a fázisban. Ekkor létrehozunk egy gráfot, amelynek csúcspontjai a csomópontokat jelölik, az élek pedig a két csomópont közt közvetlenül futó közös érintő hosszak átlagával egyezik meg. Majd ebben a gráfban keressük a legrövidebb olyan kört, amely az összes csomópontot tartalmazza, a legrövidebb Hamilton-kört. Tehát az újonnan konstruált gráfban TSP megoldása segítségével meghatározzuk a csomópontok bejárási sorrendjét, amely megadja a Σ permutációt.

A [6] itt figyelembe veszi az egyes csomópontoknál a letöltéshez szükséges úthosszat, majd ennek megfelelően ATSP-re vezeti vissza a megoldást. De a ténylegesen szükséges úthossz a látogatási kör mentén a választott érintők, érintési pontok nélkül nem számítható ki, így ez csak az adatletöltéshez szükséges úthossz, nem a robot által majd ténylegesen megtett pálya, így csak egy additív konstansként jelenik meg minden élhossznál. Ez oknál fogva az érintő hosszak átlagát veszem csak figyelembe.

Az utazó ügynök problémát lineáris programozási feladat segítségével oldom meg [16] [17].

3.2.3. Az akadályok újfajta kezelése, egyszerűsített érintőgráf létrehozása

Az **Algoritmus-1**– SVPP algoritmus az akadályok permutációba és érintőgráfba való beillesztésénél az alábbi két problémára figyeltem fel, ezekre keresek megoldást. A 3.2 ábra alapján mutatom be ezeket.

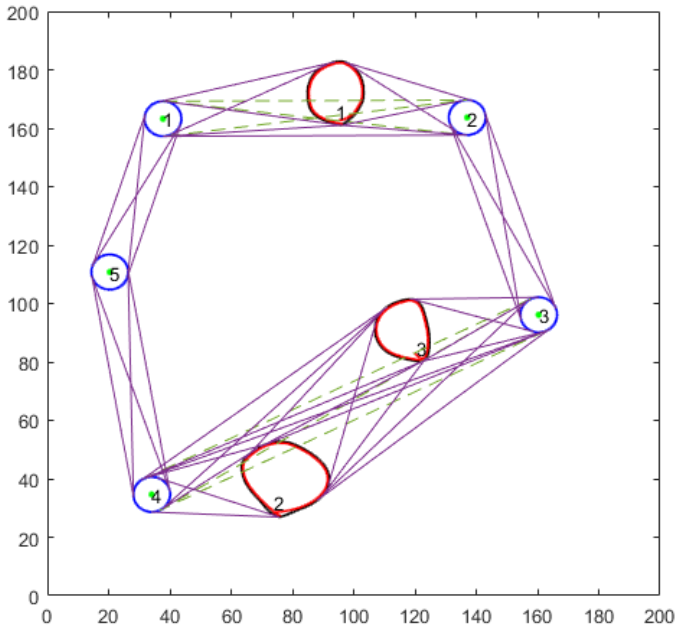
1. A 3.2 ábrán az 1. és 2. csomópont között csak egy érintő fut közvetlenül. Ha az 1. akadályt bevesszük a sorrendbe, akkor azzal kizárjuk a közvetlenül futó élt az útvonal kereséséből, amikor az rövidebb útvonalat biztosíthat, mint az akadályon keresztül. Abban az esetben azonban, ha az 1. akadályt nem vesszük be a sorrendbe, előfordulhat, hogy nem lesz megoldás az irányítottsági kényszer miatt.

2. A 3.2 ábrán a 2. és a 3. akadály különböző érintőket blokkol a 3. és 4. csomópont között. Az SVPP algoritmus feltehetően vagy az egyiket, vagy a másikat vagy mindkettőt venné be a sorrendbe, amely szintén hosszabb útvonalat eredményezne.

Az élek irányítottsága: Ahhoz, hogy az akadályok bevételéhez használt algoritmust, és majd később a fagráf elkészítését leírjam, először definiálnom kell az érintők irányítottságát. Pozitív-negatívnak (pn) nevezzük az érintő irányát, ha a kezdő objektumon a robot pozitív irányban – óramutató járásával megegyező – halad körbe, azonban az élen végig haladva, majd azt elhagyva a robot negatív irányban tud körbe haladni a következő objektum kerületén az irányítottsági kényszert betartva. Hasonló módon definiálható a pozitív-pozitív (pp), negatív-pozitív (np) és a negatív-negatív (nn) irány.

A fent említett problémák megoldására az alapötlet az, hogy nem csak egy permutációt készítek el az akadályok bevételénél hanem többet, és ezen permutációk mindegyikét felhasználom majd az egy egyszerűsített érintőgráf $G'(V', E')$ és majd a fagráf létrehozásához is. Ezt a későbbiekben **P2** feltételezésnek fogom nevezni, míg az egy permutációs esetet **P1**-nek. Tehát az új algoritmus az **Algoritmus-1**-SVPP második és harmadik lépése helyett használandó.

Az új algoritmust az **Algoritmus 2**- Akadályok hozzáadása a permutációkhoz foglaltam össze röviden. Az első lépés során készítek négy másolatot a csomópontok sorrendjét



3.2. ábra. Példa az akadályok olyan elhelyezkedésére, amelynél az SVPP algoritmus nem add feltétlenül optimális megoldást. A blokkolt érintőket szaggatott vonallal ábrázoltam.

leíró Σ permutációból, mind a négy érintő irányhoz. Ezt követően minden egymást követő csomóponthoz meghatározzuk a köztük húzott érintők által metszett akadályokat mind a négy érintőre, és az érintő irányoknak megfelelő permutációba elhelyezzük ezeket a megfelelő pozícióba. Amennyiben több akadályt is metsz ugyanaz az él, a kezdő csomóponttól való távolság alapján határozom meg a permutációban a helyüket. Ha a második lépés után azonos permutációk keletkeztek, azokból egyet megtartunk a maradékot pedig töröljük. Ezt követően ismételjük az algoritmust addig amíg még találunk új blokkoló akadályt. Az ismétlést azzal a különbséggel hajtjuk végre, hogy az első lépésben Σ helyett az összes Σ' permutációra végezzük el az algoritmust.

Algoritmus 2- Akadályok hozzáadása a permutációkhoz

1. Négy másolat készítése a Σ csomópontok permutációjából.
2. Minden egymás követő két csomópont között meghatározandóak a blokkoló akadályok mind a négy érintőre. Ezt követően az akadályok bevétele a megfelelő permutáció megfelelő helyére a blokkolt érintő iránya, illetve a kezdő csomóponttól való távolsága alapján.
3. Az egyforma permutációk közül egy megtartása, a többi törlése.
4. Ugrás az **1.** lépésre majd ismételjük az algoritmust addig amíg találunk új blokkoló akadályt. Ilyenkor Σ helyett az összes Σ' permutációt használjuk.

A fenti két esetben bevesszük az akadályok burkára illeszkedő érintőket és érintési pontokat az egyszerűsített érintőgráfba, azonban a csomópontok közötti élt se vetjük el. Ha egymás után több akadályt is metsz ugyanaz az érintő, ezek mindegyikét bevesszük a gráfba és ha nem blokkoltak, akkor mind a két szomszédos csomóponthoz húzott érintőt és természetesen a hozzájuk tartozó érintési pontokat is. Ebben az esetben az akadályok közötti érintőket is meghatározzuk és azokat is felvesszük az egyszerűsített érintőgráfba.

Az egyszerűsített érintőgráf $G'(V', E')$ tartalmazza az összes olyan érintési pontot, amelyet bármely sorrendben egymásutáni objektumok közt húzott érintők hoznak létre, és emellett tartalmazza az ezek között a pontok között értelmezett összes érintőt és ívet.

3.2.4. A fagráf elkészítése

A Dubins-jármű az útja során, vagy érintőn halad, vagy az adott objektum biztonsági sávjának külső kerületén. Az akadályoknál ez a távolság az érkezési és az elhagyási érintési pont közti út a kerület mentén, míg a csomópontoknál a látogatási kör mentén kell megtenni a kiszámolt távolságot (2.12) alapján. Az irányítottsági kényszer miatt az is könnyen belátható, hogy ha az adott objektum mentén meghatározásra került a körüljárási irány, akkor az objektum elhagyására már csak két érintő áll rendelkezésre, és ilyen módon a következő objektumhoz is már csak két érintőn tud megérkezni. Mindez természetesen csak akkor igaz, ha a meghatározott Σ' sorrendet szigorúan vesszük, tehát érintők csak az ott meghatározott sorrendben egymást követő objektumok közt futhatnak, tehát a [6] irodalomban leírtak szerint. Azonban ha több permutáció áll rendelkezésre, az akadályoknál engedékenyebben viselkedünk, akkor természetesen egy objektumból egy bizonyos körüljárási irányból kiindulva egy adott irányítottságban, lehet hogy több élen is el tudja hagyni az objektumot, nem csak két ilyen lehetőség lesz. Hiszen mehet a különböző permutációk szerinti következő objektumokhoz, amely mindegyikéhez $1, 2$ él vezethet egy adott irányítottságot tekintve. Azonban az fontos, hogy az irányítottsági feltételnek mindig megfelelően járjunk el. Az új algoritmussal tudjuk csökkenteni az út hosszát, azonban a fagráf összetettebb lesz ezáltal több lehetséges útvonal közül kell kiválasztani a legrövidebbet.

A fagráf létrehozását és abban legrövidebb út keresését a [6] irodalom javasolja, azonban az alábbi két, **Algoritmus 3** és **Algoritmus 4** algoritmusokat dolgoztam ki a fagráf elkészítésének implementálására.

Algoritmus 3- Fagráf létrehozása egy Σ' permutáció esetén

1. Kezdő és végpontok felvétele csúcsként a $T(V, E)$ fagráfba, mind negatív, mind pozitív irányítottság szerint.
2. Objektumonként a 4 érkezési és 4 elhagyási érintési pont felvétele csúcsként a $T(V, E)$ fagráfba, figyelembe véve a Σ' permutációt.
3. Az egyes objektumokon megtett ívhosszak kiszámítása minden lehetséges (irányítottsági kényszert betartó) beérkezési és elhagyási pontpár között, ügyelve a csomópontok és akadályok megkülönböztetésére. Ezek felvétele élként E-be a $T(V, E)$ megfelelő csúcsait összekötve.
4.Az összes a Σ' permutációban egymást követő objektum szerint az érintők felvétele élként a $T(V, E)$ megfelelő csúcsai közé ügyelve az irányítottsági kényszer betartására.
5. A bázis csomóponton a kezdő pont és az egyes indulási és érkezési pontok közötti ívhossz kiszámítása, ezek felvétele élként E-be a $T(V, E)$ megfelelő csúcsait összekötve.

A fagráf elkészítését abban az esetben ha csak egy Σ' permutáció áll rendelkezésre **P1** az **Algoritmus 3** foglalja össze, míg a több Σ' permutációs esetet **P2** az **Algoritmus 4**. A fagráf elkészítése során először létrehozok két kezdő és két vég csúcspontot, amelyek a különböző körüljárási irányból induláshoz – és mivel a robot körbe-körbe halad így befejezéshez – szükséges. Ezek elnevezése `Start_n` és `End_n` a negatív, illetve `Start_p` és `End_p` a pozitív körüljáráshoz. Ezt követően sorba a Σ permutáció szerint beveszem a gráfba a csomópontokat. Először is a csomópontokhoz tartozó csúcspontokat kell létrehozni. A kimenő érintési pontban elhelyezkedő csúcsokat `Nodenév_irany` elnevezésekkel láttam el, ahol az irány az érintő iránya. Az induló érintési pontok pedig, `előzőNodenév_to_jelenlegiNodenév_irany` szerint kaptak nevet. Ezt követően a csú-

Algoritmus 4- Fagráf létrehozása több Σ' permutáció esetén

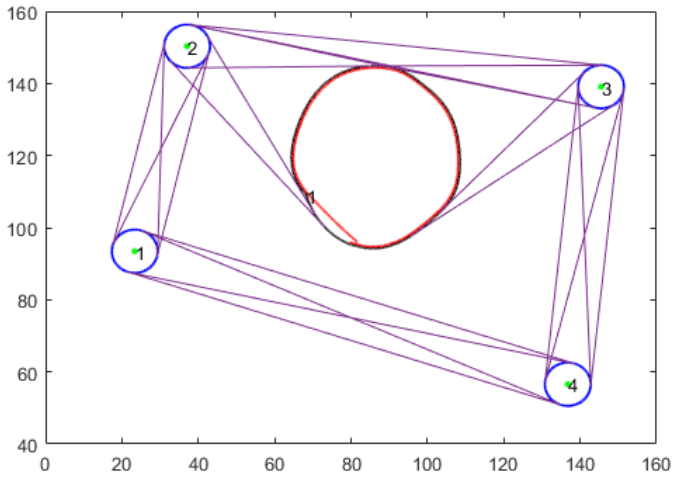
1. Algoritmus 3 elvégzése a csomópontok Σ permutációjára (TSP által meghatározott bejárási sorrend), azzal a kikötéssel, hogy csak azokat az elemeket vesszük fel a $T(V, E)$ fagráfba amelyek szerepelnek a $G'(V', E')$ egyszerűsített gráfban.

2. A Σ' permutációk mindegyikén végighaladva

1.) Ha csak egy akadály van két csomópont között az i-edik helyen az adott permutációban: **Algoritmus 3**, **2-4** lépés elvégzése a $\sigma_{i-1}, \sigma_i, \sigma_{i+1}$ hármasra (tehát az akadályt megelőző és követő csomópontra és az akadályra). Csak akkor ha a bevett élek és csúcspontok elemei $G'(V', E')$-nek.

2.) Ha több akadály van két csomópont között, ezek legyenek $\delta O = \{\delta o_1, \cdots \delta o_j\}$: 1.) lépés elvégzése egyesével az összes $\delta o_i \in \delta O$ akadályra az adott permutációban az azt megelőző és követő csomópontra. Az **Algoritmus 3**, **2-4** lépés elvégzése páronként az összes $\delta o_i \in \delta O$ akadályra –akkor is ha azok a permutációban nem egymást követő elemek– természetes csak akkor ha a csúcspontok és az élek elemei $G'(V', E')$-nek.

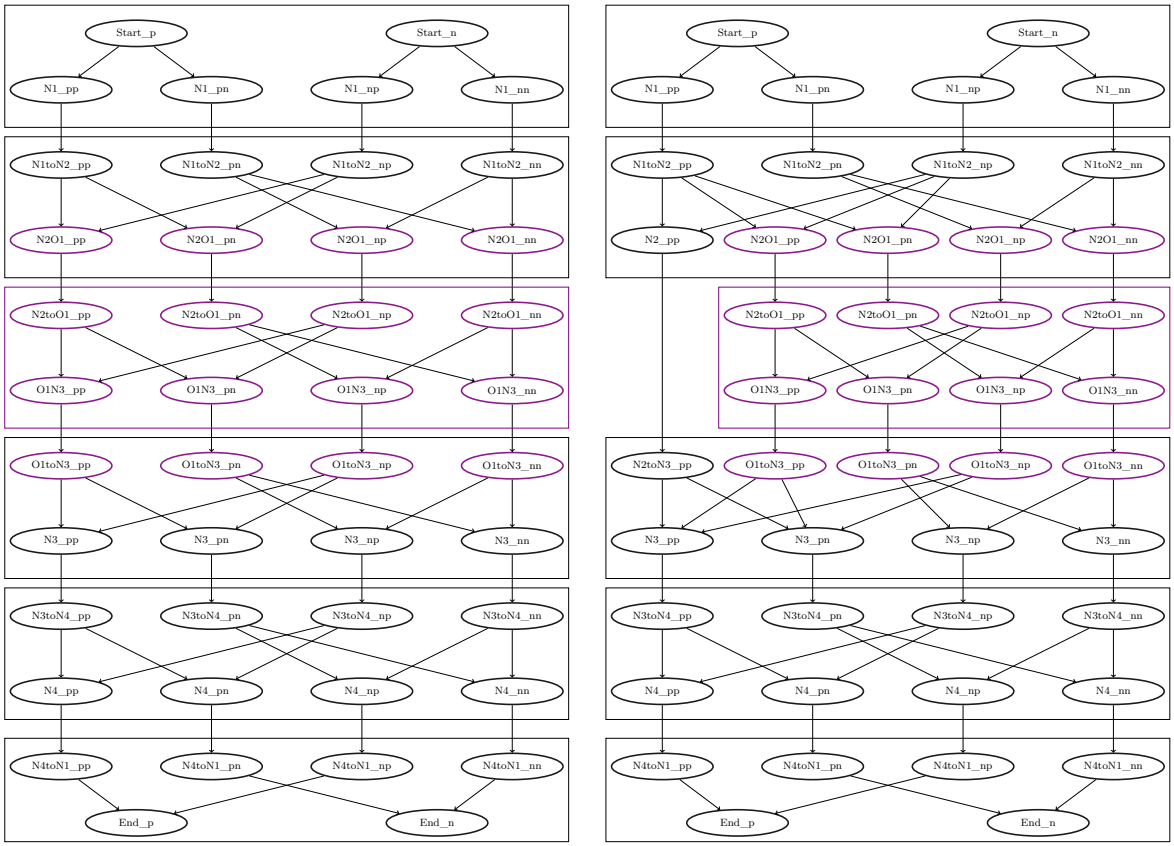
csok közötti élek meghatározása a feladat. Értelemszerűen a `Start_p` csúcsból a bázis állomás `Node_1_pp` és `Node_1_pn` csúcsába, míg a `Start_n` csúcsból a `Node_1_np` és `Node_1_nn` csúcsába húzható él, ezek hossza fogja mutatni, hogy a kezdőpontból mekkora íven lehet eljutni az egyes kimeneti érintési pontokhoz. A kimeneti érintési pontokból, a sorban következő érzékelő csomópont indulási érintési pontjába rajzolandó él, ezek hossza, maga az irányítottságnak megfelelő érintőhossz. Ezt követően ezekből a pontokból kell meghatározni az adott érzékelő csomópont körüli ívhosszakat a kimeneti érintési pontoknak és a bemeneti irányítottságoknak megfelelően az irányítottsági kényszert betartva. Ezt a permutációban szereplő sorrendben az összes csomópontra el kell végezni, majd az utolsó elemet követően, abból az első, bázis csomópontra fel kell venni a `Node_end_to_Node_1_irany` csúcspontokat, és a kimeneti érintőpontokhoz tartozó ívhosszak élét, majd a helyes irányítottságnak megfelelően be kell venni az éleket a `End_n` és `End_p` csúcspontokba.



3.3. ábra. Példa elrendezés a fa-gráf létrehozásának bemutatásához, a csomópontok permutációja, $\Sigma = \{C_1, C_2, C_3, C_4, C_5\}$

Ezt követően az akadályok bevétele következik a Σ' permutációk szerint. Mivel egy akadály több csomópont közötti élt is blokkolhat, annak érdekében, hogy a legrövidebb út megkeresése során egyik csomópont se maradjon ki az útból, az akadályok aszerint

nevezem el, hogy melyik két csomópont közti utat blokkolják, így ha egy akadály több csomópont közötti élt is blokkol, külön elnevezésben részesülnek, így nem fordulhat elő olyan eset, hogy az algoritmus egy olyan legrövidebb utat talál meg, amely egy-egy csomópontot kihagy. Tegyük fel, hogy az egyes sorszámú `Obstacle_1` akadály a `Node_2` és `Node_3` közötti utat blokkolja a 3.3 ábra szerinti elrendezés. Ekkor először is fel kell venni a `Node_2` látogatási körén azokat az elhagyási pontokat amelyek az `Obstacle_1` akadályba futó érintők kezdő pontjai. Ezeket nevezzük `Node_2_Obstacle_1_irany` csúcspontoknak. Emellett fel kell venni az akadályon a beérkezési pontokat, majd az innen induló `Node_3` csomópontba futó érintők elhagyási pontjaihoz tartozó csúcspontokat. Végezetül pedig a `Node_3` csomópont látogatási körén elhelyezkedő beérkezési érintési pontokhoz tartozó csúcspontokra is szükség van. Ezek között hasonlóan a fent leírtakhoz éleket kell definiálni, arra ügyelve, hogy az irányítottságok helyesek legyenek.



3.4. ábra. Fagráf a 3.3 ábra példa elrendezéshez **Algoritmus 3** és **Algoritmus 4** használatával, az akadályok lilával kerültek ábrázolásra.

Előfordulhat, hogy egymást kettő vagy több akadály követ. Ebben az esetben ha létezik érintő az akadályok alkotta blokk előtti és utáni csomópontba, akkor azt minden esetben felveszem a fagráfba, illetve az akadályok közötti érintőket is a fent leírtak szerint. Ebben az esetben a csúcsok elnevezésében mindig az az objektum neve áll elől amely hamarabb szerepel a sorrendben. Hasonlóan lehet eljárni akkor is ha a kezdő csomópont előtt vagy után van akadály, ilyenkor arra kell ügyelni, hogy az ívhosszat a `StartPont`-ból vagy abba beérkezően kell számolni.

Az eredeti SVPP algoritmus szerinti fagráf létrehozáskor úgy járhatunk el az akadályok bevételekor mintha csomópontok lennének. Egyszerűen a kialakult Σ' permutáción

végighaladva az összes csomópont és akadály érintési pontjait felvesszük csúcsoknak a fagráfba, majd köztük az irányítottsági kényszernek megfelelően berajzoljuk az íveket és érintőket jelképező éleket.

A példa szerinti fagráfot mind az **Algoritmus 3** és **Algoritmus 4** algoritmusokhoz a a 3.4 ábra mutatja, ami a 3.3 ábra elrendezéshez készült. Az **Algoritmus 4** segítségével elkészített gráf létrehozása során a fent leírtaknak megfelelően először a feketével jelölt csomópontokhoz tartozó csúcsokat vettem be a gráfba, majd az ezekhez tartozó éleket. Ezt követően a lilával jelölt akadályokhoz tartozócsúcsok és élek kerültek sorra. Az ábrán az egyes objektumokhoz tartozó csúcsokat külön téglalapba foglalva ábrázolom.

3.2.5. A legrövidebb út megkeresése a fagráfban

A fagráf elkészítése után a legrövidebb út megtalálása könnyen elvégezhető a különböző gráfbejárási algoritmusok segítségével, a megvalósítás során Dijkstra algoritmust [9] alkalmaztam. Érdemes azonban a keletkező utat megvizsgálni úgy, hogy a bázis állomáson két különböző irányba – óramutató járásával egyező és ellentétes– indítjuk el a robotot. Illetve a Σ' permutáción fordított sorrendben végig haladva is meghatározzuk a két kezdeti irányra a legrövidebb útvonalat, és az így keletkező négy eredményből választjuk ki a legrövidebbet.

4. fejezet

Több robottal való bejárás

A kisebb késleltetési idő elérése érdekében a már bemutatott feladatot több egymással együttműködő robotra is megoldom. Ilyenkor a lényeg az, hogy a különböző robotok különböző érzékelő csomópontokat járjanak be úgy, hogy mindnyájan ugyanabból a bázis csomópontból teszik meg az útjukat. A cél az egyes robotok bejárási idejének minimalizálása mellett, hogy megközelítőleg azonos idő alatt végezzék el a feladatukat, annak érdekében, hogy az összes érzékelő csomópontról gyűjtött adat megközelítőleg azonos késleltetéssel érkezzen be a bázis állomásra.

A feladat megoldására három különböző lehetőséget mutatok be, majd ezek hatékonyságát hasonlítom össze, az összbejárási idő, illetve a különböző robotok bejárási idejének vizsgálatával.

Az algoritmusok elnevezésében a "k" betű arra utal, hogy a feladatot k darab robot végzi el együttesen, így ennyi alútvonal készítendő.

A következő algoritmusokban szereplő SVPP algoritmus helyett használható az általam módosított M-SVPP a 3.2 fejezetben leírtak alapján, beleértve az **L2** és **P2** feltételezéseket is.

A k-SPLITOUR és k-SVPP algoritmusok felvázolásra kerültek a [6] irodalomban. A következő, k-means algoritmust használó algoritmust azonban én készítettem el.

4.1. A feladat megoldása k-means algoritmus segítségével

A k-means algoritmus [11] egy adott ponthalmaz elemeit úgy osztja k darab csoportra, hogy a klasztereken belül a pontok euklideszi távolsága minimális legyen. A feladat megoldása során ennek felhasználását az **Algoritmus 5** foglalja össze. A csomópontokra a bázis csomópont kivételével elvégzem a k-means algoritmust. Minden klaszterbe beveszem a bázis csomópontot (C_1). Az így keletkező klaszterekre egyesével elvégzem az SVPP algoritmust.

Algoritmus 5–k-means alkalmazása a feladatra
1. k-means elvégzése $C_1 \cdots C_n$ csomópontokra.$\longrightarrow Cluster_1, \cdots, Cluster_k$
2. Minden $Cluster_l$-hoz, $l \in [1, k]$ a C_1 bázis csomópont hozzáadása
3. SVPP algoritmus elvégzése minden $Cluster_l$-re, $l \in [1, k]$ és így P_l meghatározása.

Fontosnak tartom kiemelni, hogy a k-means algoritmus véletlen helyről indul, tehát különböző végrehajtáskor eltérő eredményt kaphatunk. Így az **Algoritmus 5** sem ad egyértelmű megoldást egy terepen, ezért javasolt a többszöri elvégzése és abból a legjobb eredmény kiválasztása.

4.2. A feladat megvalósítása k-SPLITOUR algoritmussal

A k-SPLITOUR algoritmus [10] az MTSP, vagyis a több kereskedős utazó ügynök problémára kínál megoldást. A cél az, hogy a különböző kereskedők, megközelítőleg azonos hosszúságú útvonalat járjanak be, úgy, hogy közben minimális legyen az út hossza. Az [6] a k-SPLITOUR algoritmust az aktuális feladat megoldásához igazította, ennek összegfoglalója az **Algoritmus 6**. Első lépésként meghatározza egy robotra az útvonalat a SVPP algoritmus segítségével. Majd az útvonal hossza L, illetve a bázis csomópont és attól a legtávolabbi csomópont távolságának L_{max} és a P egy robotos útvonal segítségével meghatározza a csomópontok halmazait a különböző robotokra. A P útvonalon a csomópontok permutációja $\Sigma = \{\sigma_1, \sigma_2, \cdots, \sigma_n\}$. A $len(\sigma_1, \sigma_i)$ $i \in [2, n]$ függvény kiszámítja a távolságot a σ_1 és σ_i látogatási körök távolságát a P mentén. Tehát ha két látogatási kör között P mentén akadály is elhelyezkedik, akkor a P-ben meghatározott érintők az akadályhoz a csomópontokból és az akadályon megtett ívhossz illetve a későbbi látogatási körön megtett ívhossz összege lesz a köztük lévő úthossz.

Algoritmus 6–k-SPLITOUR alkalmazása a feladatra

1. SVPP algoritmus elvégzése, az így talált útvonal P, hossza $|P| = L$
2. Minden l-re, $l \in [1, k-1]$, meg keressük az utolsó látogatási kört P-ben, amelyre még teljesül, hogy $len(\sigma_1, \sigma_{i(l)}) \leqslant \frac{l}{k}(L - 2L_{max}) + L_{max}$, így a következő klaszterek írhatók fel:
$Cluster_1 = \{\sigma_1, \sigma_2, \cdots, \sigma_{i(1)}\}$
$Cluster_2 = \{\sigma_1, \sigma_{i(l-1)+1}, \cdots, \sigma_{i(l)}\}$
$Cluster_k = \{\sigma_1, \sigma_{i(k-1)+1}, \cdots, \sigma_n\}$
ahol a σ alsó indexe a látogatási körök a P-ben való helyüket mutatja.

A k-SPLITOUR algoritmusban létrejött klaszterek egyúttal a klaszterekbe tartozó látogatási körök bejárási sorrendjét is meghatározza. Az algoritmus lényegében k darab részre vágja szét az egy robotra meghatározott útvonalat.

Ha a P útvonalon a bázis csomópontra sok adat kerül feltöltésre a robotnak sok kört kell tennie, így az L útvonal egy nem elhanyagolható részét ez jelenti. A klaszterek meghatározását azonban ez elronthatja, a második lépésben feltüntetett egyenlőtlenség későbbi csomópontokra is teljesül, és ennek hatására az utolsó robotoknak kisebb bejárandó úthossz marad. Hiszen a bázis csomópont körbejárását minden robotnak végre kell hajtania, de ezt az útvonalat ilyen módon csak az utolsó robot útvonalához számítjuk hozzá.

A fent leírtak miatt, a k-SPILTOUR algoritmus elvégzéséhez a munkámban L helyett $L' = L - l_B = L - l_B$ úthosszat használom, ahol l_B a bázisállomás látogatási körén való megtett úthosszat jelöli az egy robotra tervezett útvonal végén.

4.3. A k-SVPP algoritmus

A k-SVPP algoritmus [6] irodalomban valósították meg, az **Algoritmus 7**–kSVPP foglaltam össze röviden. Az első két lépése megegyezik a k-SPLITOUR algoritmussal, harmadik lépéseként, pedig minden klaszterre elvégzi az SVPP algoritmust a rövidebb útvonal elérése érdekében. Várhatóan ez az algoritmus jobb eredményt ad mint a k-SPLITOUR, hiszen itt, az ott meghatározott klaszterekben szereplő elemeket esetlegesen jobb sorrendbe rendezi, így csökkentve a bejárandó útvonalat. Ugyanakkor az egy robotra kapott megoldás szétdarabolása –k-SPLITOUR algoritmus– pedig közel azonos időtartamú bejárást biztosít a klaszterekre.

Az SVPP algoritmus helyett, ahogy az előző kettő algoritmus során is M-SVPP algoritmust használok. A k-SPLITOUR algoritmust az előző fejezetben megfontoltak alapján L' útvonal hosszat feltételezve hajtom végre.

Algoritmus 7–k-SVPP

1. SVPP algoritmus elvégzése, az így talált útvonal P, hossza $|P| = L$

2. Minden l-re, $l \in [1, k-1]$, meg keressük az utolsó látogatási kört P-ben, amelyre még teljesül, hogy $len(\sigma_1, \sigma_{i(l)}) \leqslant \frac{l}{k}(L - 2L_{max}) + L_{max}$, így a következő klaszterek írhatók fel:
$Cluster_1 = \left\{\sigma_1, \sigma_2, \cdots, \sigma_{i(1)}\right\}$
$Cluster_2 = \left\{\sigma_1, \sigma_{i(l-1)+1}, \cdots, \sigma_{i(l)}\right\}$
$Cluster_k = \left\{\sigma_1, \sigma_{i(k-1)+1}, \cdots, \sigma_n\right\}$
ahol a σ alsó indexe a látogatási körök P-ben való helyüket mutatja.

3. SVPP algoritmus elvégzése minden $Cluster_l$-re, $l \in [1, k]$ és így P_l meghatározása.

A k-SPLITOUR algoritmussal ellentétben itt az egy robotra létrehozott látogatási körök bejárási sorrendje felbomlik az egyes klaszterekben. Ezáltal az egyes klaszterekben külön-külön közel optimális a bejárás. Azonban ez nem jelenti azt, hogy az egész feladat megoldása is az.

5. fejezet

Szimulációs eredmények

Először bemutatom a pálya előkészítésének menetét, majd egy robot használatával a pályatervezés lépéseit, az általam tett feltételezéseket összehasonlítva a [6] irodalomban tettekkel, mint az akadályok kezelése a bejárási sorrendben, és a látogatási körökön átmenő érintők használata. Ezt követően a több robotot alkalmazó algoritmusokat mutatom be, majd össze hasonlítom ezeket. Az algoritmusok jóságát a használt robotok számának a függvényében is megvizsgálom mind a három használt algoritmusra. A több robotos algoritmusoknál mind az akadályok bevételét, mind a látogatási körön átmenő érintők esetében az általam tett feltételezésekkel élek, ezekkel mutatom be az algoritmusokat.

5.1. A terep előkészítése, útvonal tervezés egy robot számára

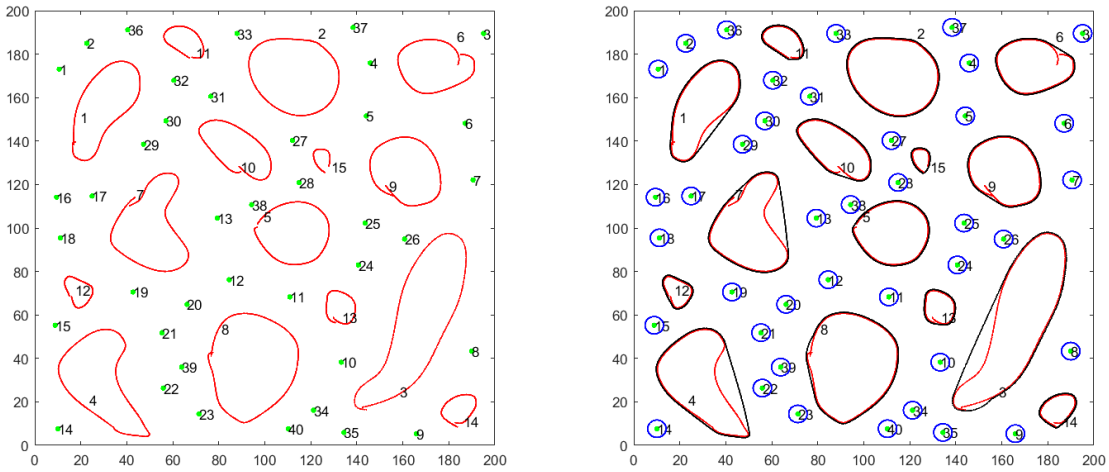
A terep amelyről adatokat szeretnénk gyűjteni az 5.1 ábra első képén látható. Ez egy $200m \times 200m$ nagyságú terület, amelyen egy darab bázis állomás és 39 darab érzékelő csomópont található. Minden érzékelő csomópont $g = 0.5MB$ adatot tárol, amelyet a robotnak le kell tölteni, és ennek megfelelően a bázis csomóponton $g_B = (n-1)\,0.5MB = 19.5MB$ adatot tölt fel későbbi feldolgozásra. A csomópontok le- és feltöltési sebessége $r = 250kB/s$ a látogatási körök mentén. A pálya bejárását nehezíti a 15 darab akadály a terepen.

Ha a terep és az azon elhelyezkedő objektumok rendelkezésre állnak a következő feladat a biztonsági sávok, látogatási körök felvétele a bejáró robot tulajdonságait figyelembe véve. A robot konstans $v = 4\,m/s$ sebességgel halad, és a maximális szögsebessége $|u_M| \leq 1\,rad/s$. A robot minimális fordulási, és a látogatási körök sugara $R_{min} = \frac{v}{u_M} = 4\,m$. Az objektumoktól a robotnak minimum $d_{safe} = 0.5\,m$ távolságra kell elhaladnia, a sérülés elkerülése érdekében. Az akadályok mentén a biztonsági burkoló görbét az akadályok konvex burkától számítom a 2.3 fejezetben megfontoltak miatt. A burkolókkal ellátott akadályokat és a csomópontok köré húzott látogatási köröket az 5.1 ábra második képe mutatja.

A látogatási körök és akadály burkolók meghatározása után a következő lépés a $G(V, E)$ érintőgráf létrehozása, tehát az érintési pontok és érintők meghatározása az objektumok közt, majd a **L1** vagy **L2** feltételnek megfelelően az akadályok burkoló görbéjét és a látogatási köröket, vagy csak az akadályok burkoló görbéjét metsző érintők elvetése. Az 5.2 ábra szemlélteti a két feltétel lényegi eltérését.

Az 5.2 ábrán észrevehető, hogy az én általam használt feltételezés **L2** sokkal több érintőt vesz be a $G(V, E)$ érintőgráfba, ezáltal növelve a lehetséges útvonalak számát, és ugyanakkor a számítás kapacitás igényt.

Következő lépés a csomópontok bejárási sorrendjének meghatározása Σ, majd az akadályok bevétele a permutációba **P1** vagy permutációkba **P2** és így Σ' permutáció és



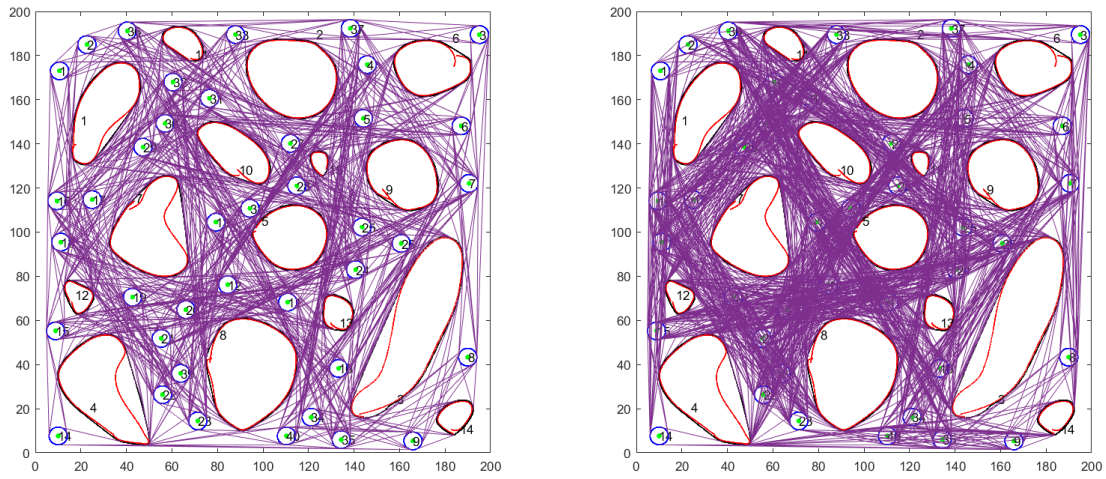
5.1. ábra. A bejárandó terep. A csomópontok zölddel, az akadályok pirossal jelölve szerepelnek. Az objektumok mellett láthatók a sorszámaik, amelyek a megoldás elemzését segítik majd. A burkoló görbék (fekete) és látogatási körök (kék) a bejárandó terepen.

$G'(V', E')$ egyszerűsített érintőgráf elkészítése. Az egyszerűsített érintőgráfot az 5.3 ábrán szemléltetem. Az első ábrán a már **L1** feltételezéssel létrehozott $G(V, E)$ érintő gráfot egyszerűsítem az **P1** betartásával, egy permutációs eset. A második ábrán pedig a már **L2** feltételezéssel létrehozott $G(V, E)$ érintőgráfot egyszerűsítem a **P2** alkalmazásával, több permutációs eset.

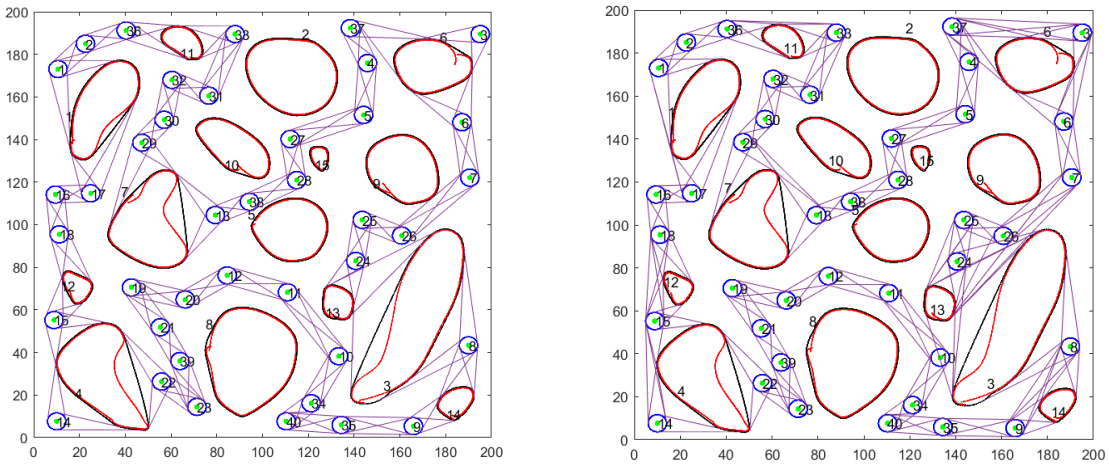
Az 5.3 ábrán észrevehető, hogy, abban az esetben, amikor a látogatási köröket is metszhetnek érintők és az akadályok bevételéhez **Algoritmus-2** kerül felhasználásra, **L2-P2** eset, sokkal több érintő áll rendelkezésre az egyszerűsített érintőgráfban, és így majd a fagráfban is. Vagyis ennek köszönhetően nő a bejárható útvonalak száma, a legrövidebb út keresési algoritmus sikeresebb lehet. Azonban fontos megjegyezni, hogy komplexebb fagráf keletkezik. A másik esetben, vagyis amikor azok az érintők sem szerepelhetnek az érintőgráfban, amelyek látogatási köröket metszenek, és az objektumok sorrendjére csak egy permutáció áll rendelkezésre, **L1-P1** eset, egy objektumba legfeljebb négy érintő fut be és legfeljebb négy érintő indulhat ki belőle. Általában négy-négy érintő fut be és ki, de például a 13. és 3. akadály között csak három használható él van. Észrevehető ugyanakkor, hogy a **L2-P2** esetben a 13. és a 3. akadály közt négy érintő fut, amelyből az egyik metszi a 10. csomópont látogatási körét. Az **L2-P2** esetben a 37. és 3. csomópont közt fut három közvetlen él, míg a **L1-P1** esetben csak a 6. akadály közbeiktatásával tud a robot majd közlekedni.

Ezt követően a T fagráf létrehozása következik a **P1** vagy **P2** feltételezésnek megfelelő **Algoritmus 3** és **Algoritmus 4** alapján. A bázis csomópont az 1-es sorszámmal jelölt. A fagráfban mind **L1-P1** mind **L2-P2** esetre, mind pozitív mind negatív kezdő irányra megkeresem a legrövidebb útvonalat a megfelelő T fagráfban.

Az 5.4 és 5.5 ábra szerint az **L2-P2** feltételezéssekkel élve rövidebb a bejárandó útvonal hossza. Azt is észrevehetjük hogy az egyes feltételezésekkel élve a negatív és a pozitív kezdő irányú utak nem térnek el csak a kezdő csomóponthoz közvetlenül csatlakozó érintőben. A jobb vizuális vizsgálat érdekében a robotok által megtett útvonal kirajzolása során a látogatási körök mentén megtett útvonalat is ábrázoltam. A **L2-P2** feltételezésekkel megvalósított útvonal tervezéskor észrevehető, hogy a 24. csomópont és a 3. akadály

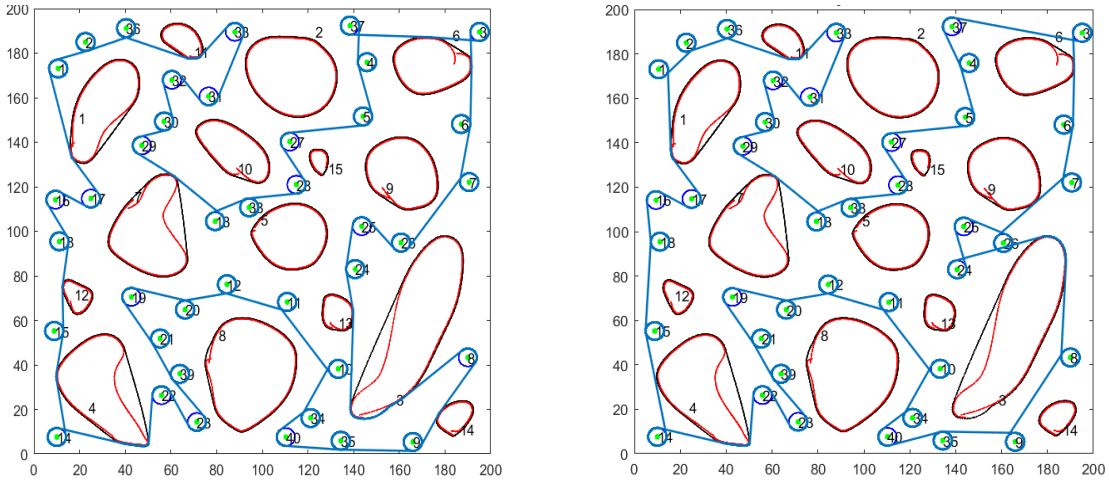


5.2. ábra. Az $G(V, E)$ érintőgráf **L1** és **L2** feltételezésekkel élve.

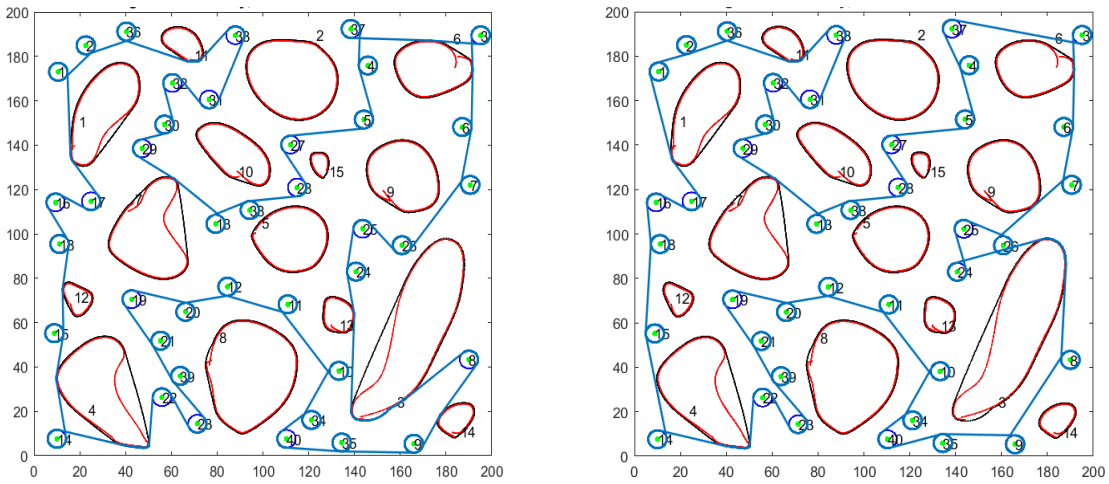


5.3. ábra. Az $G'(V', E')$ egyszerűsített érintőgráf **L1** és **P1** illetve **L2** és **P2** feltételezésekkel élve.

közötti érintő metszi a 26. csomópont látogatási kört, tehát ez az érintő is szükséges a ahhoz, hogy a lehető legrövidebb útvonalat tervezhessünk. A 37. és a 3. csomópont közt az **L1-P1** esetben a 6. akadályon keresztül vezet az út, ezzel szemben a **L2-P2** esetben úgy vezet közvetlen él, hogy a 37. csomóponton megtett ívhossz lényegesen kevesebb mint az előző esetben. A fent megadott adatok alapján az egyes csomópontokon a robotnak $l = \frac{g\,v}{r} = 8m$ ívhosszat kell legalább bejárnia, hogy az összes adatot le tudja tölteni, ez megközelítőleg a látogatási kör kerületének egy harmada. Észrevehető, hogy például a 32-31-33 csomópontok közt a robot olyan íven halad, hogy ne kelljen teljest kört megtennie az egyes látogatási körök mentén.



5.4. ábra. A T fagráfban talált legrövidebb pozitív kezdőirányú út **L1** és **P1** illetve **L2** és **P2** feltételezésekkel élve. A tervezett út hossza rendre 2337.6 m és 2287.7 m

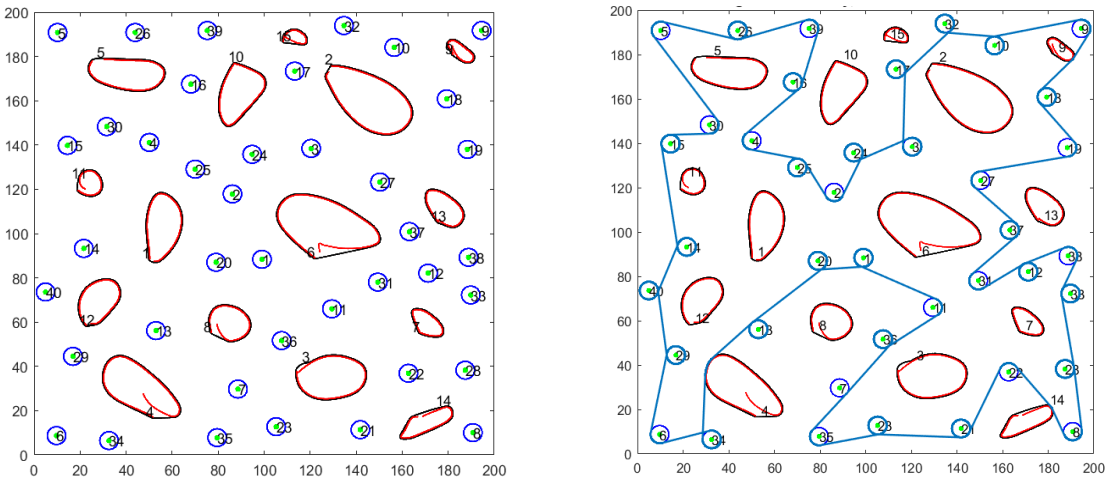


5.5. ábra. A T fagráfban talált legrövidebb negatív kezdőirányú út **L1** és **P1** illetve **L2** és **P2** feltételezésekkel élve. A tervezett út hossza rendre 2341.9 m és 2275.1 m

5.2. Útvonal tervezés több robot számára

Az útvonal tervezést több robot számára az **Algoritmus 5-7** alapján végeztem el M-SVPP algoritmust használva. Az érintők metszhetnek látogatási köröket **L2**, és a fagráfot több permutáció segítségével konstruáltam meg **P2**. A szimulációkat $k \in [2, 10]$ robotra készítettem el mindhárom algoritmusra. Ezt követően pedig elemzem a különböző algoritmusokat a kapott alútvonalak alapján.

A bejárandó terepet és az egy robotos megoldást az 5.6 ábra mutatja. A bejárást csak negatív irányba mutatom meg, a pozitív kezdő irányú megoldás az előzőekhez hasonlóan csak a bázis csomóponthoz csatlakozó élekben tér el, az útvonalbeli különbség minimális. A több robotos megoldások során is csak a negatív kezdő irányú esetet vizsgálom. Természetesen itt a különböző robotok indulhatnának különböző irányba, megfelelő időzítéssel egymáshoz képest. Azonban a robotok mozgásának időbeli összehangolása most nem cél. Az előző fejezetben feltüntetett robot és csomópont paraméterek, mint robot haladási és szögsebesség, letöltési sebesség és adat mennyiség most is érvényes, annak kivételével, hogy a báziscsomóponton most $g_B = \frac{1}{k}(n-1)\,0.5MB$ adat kerül feltöltésre. A terepen továbbra is 40 csomópont és 15 akadály helyezkedik el.

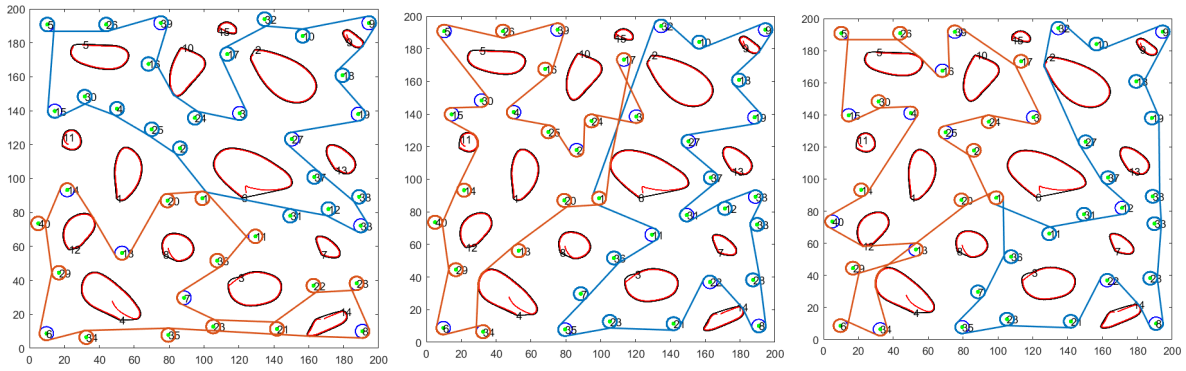


5.6. ábra. A bejárandó terep és a meghatározott negatív kezdő irányú egy darab robotra tervezett út, melynek hossza: $L = 2275.5\,m$

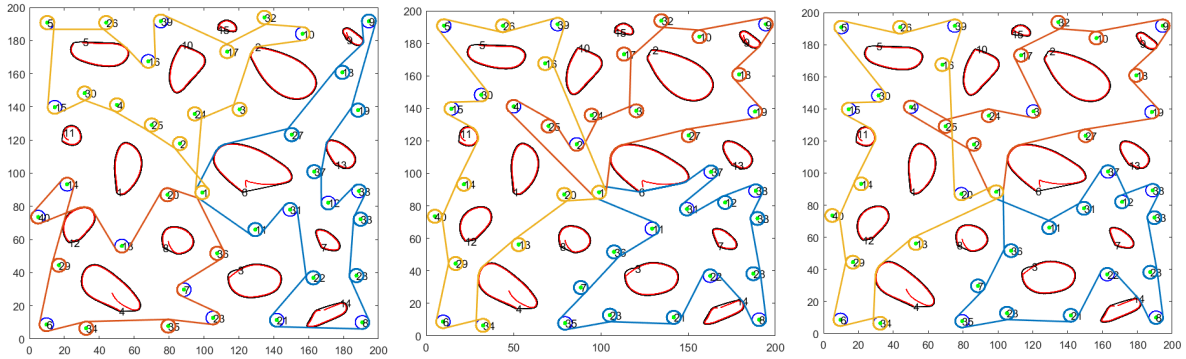
Az egy robotos megoldás során, a robotnak az útja végén $g_B = 19.5MB$ adatot kell feltöltenie amihez $l_B = \frac{g_B\, v}{r} = 312\,m$ szükséges. Ez az úthosszhoz képest nem elhanyagolható mennyiségű, ha az ezzel együtt vett útvonal hosszával határoznánk meg k-SPLITOUR algoritmus szerint a klasztereket, akkor az így keletkezett alútvonalak hosszukban jelentősen eltérnének. Ezért a k-SPLITOUR és a k-SVPP algoritmusokban nem az 5.6 ábra szerinti útvonalhosszat használom, hanem $L' = |P| - l_B = L - l_B$ a korábbi jelöléseket használva.

A $k \in [2, 10]$ robotra elvégzett útvonaltervezést k-means, k-SPLITOUR és k-SVPP algoritmusok alapján a 5.7-5.15 ábrák szemléltetik. Az egyes részútvonalak hosszát rendre a 5.1, 5.2 és 5.3 táblázatok tartalmazzák. Az alútvonalakra vonatkozó statisztikai adatokat, az alútvonalak maximális értékét, összegét és az átlagos eltérést $\frac{1}{n}\sum_{i=1}^{n}|x - \overline{x}|$ képlet szerint, a 5.16 ábra szemlélteti összehasonlítva az egyes algoritmusokat és az alkalmazott robotok számát.

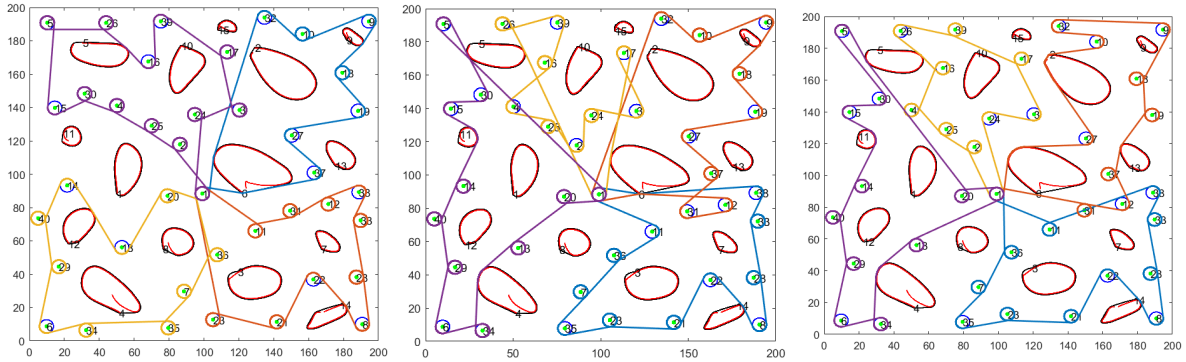
A 5.7 - 5.15 ábrákon látható, hogy a k-means algoritmus csomópontokban is más eredményt ad mint a többi. Ez az érzékelő csomópontok egymástól való távolságát mi-



5.7. ábra. A k-means, k-SPLITOUR, k-SVPP alkalmazása két robotra



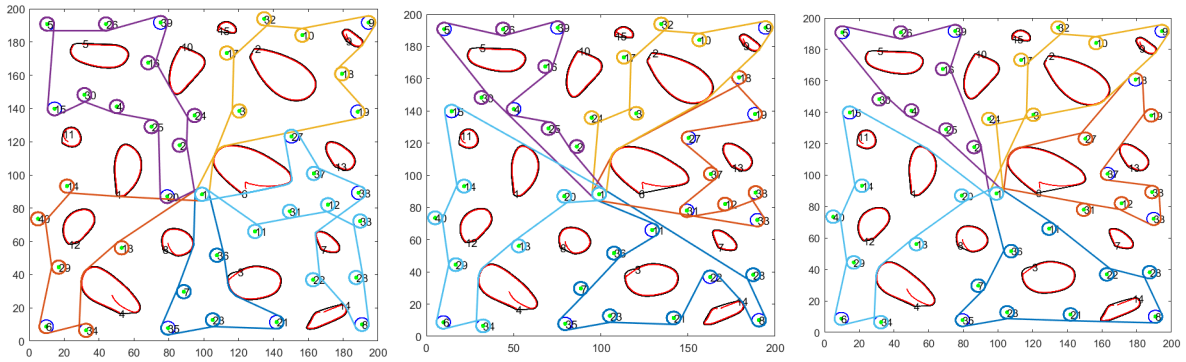
5.8. ábra. A k-means, k-SPLITOUR, k-SVPP alkalmazása három robotra



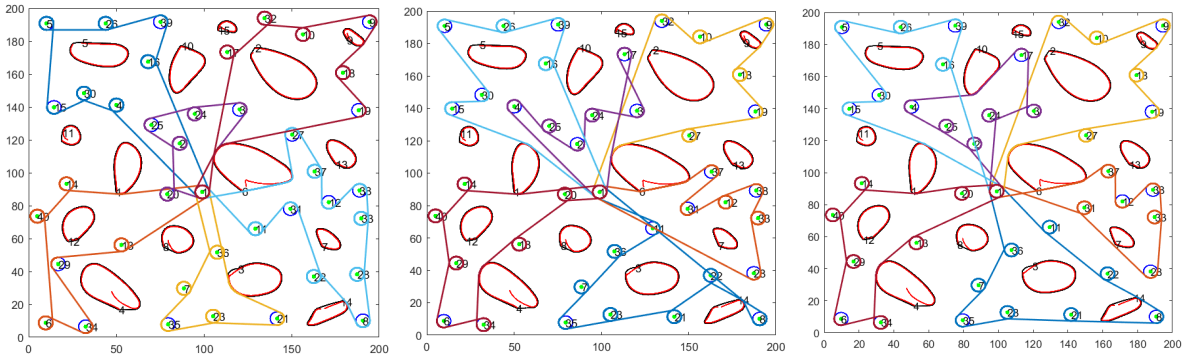
5.9. ábra. A k-means, k-SPLITOUR, k-SVPP alkalmazása négy robotra

nimalizálja egy klaszteren belül, azok meghatározásakor. A k-SPLITOUR és a k-SVPP a csomópontjaikban megegyeznek, hiszen a k-SVPP a k-SPLITOUR algoritmus klasztereit használja fel. Azonban például az 5.9 ábrán látványos, hogy a k-SVPP "kisimítja" a k-SPLITOUR-ben a vágások következtében létre jött távolságokat a klaszterek két végén.

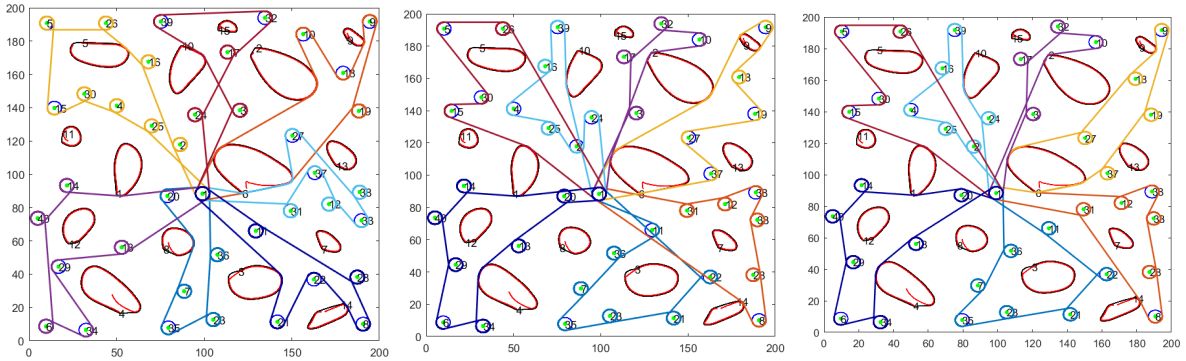
A k-means algoritmus $k \in [2, 5]$ robotot alkalmazva megközelítőleg jó megoldást ad. A klasztereket egymástól és a bázis csomóponttól is egyenletesen elhelyezve rendezi csopor-



5.10. ábra. A k-means, k-SPLITOUR, k-SVPP alkalmazása öt robotra

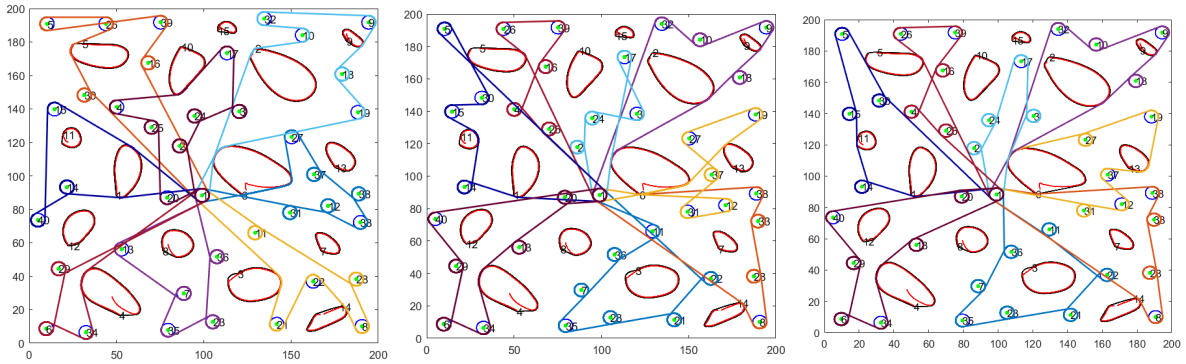


5.11. ábra. A k-means, k-SPLITOUR, k-SVPP alkalmazása hat robotra

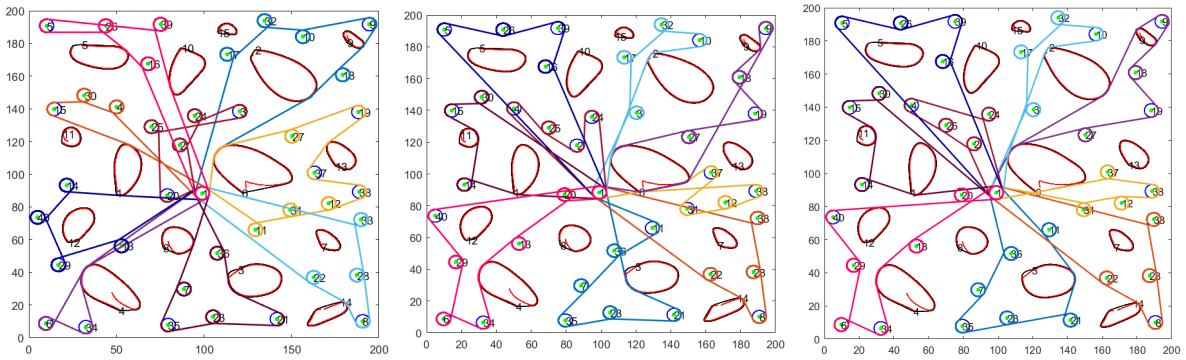


5.12. ábra. A k-means, k-SPLITOUR, k-SVPP alkalmazása hét robotra

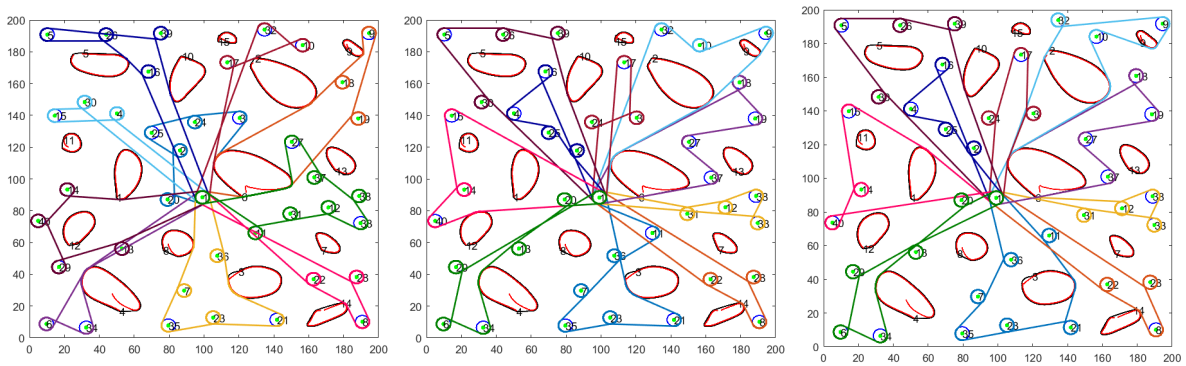
tokba. Azonban az egyes klaszterekben a bejárási alútvonalak akár $200 - 250\,m$ hosszban is eltérhetnek egymástól, amely $t = 50 - 85s$ bejárási időt jelent. A csoportokon a M-SVPP algoritmus a legtöbb esetben jó minőségű legrövidebb útvonal keresést hajt végre. A TSP megoldása során azonban néha előfordul, hogy az útvonalban hurok alakul ki a nem megfelelő optimalizálás következtében, például a 5.8 ábra első képén a 29-14-40-13 csomópontokra. A k-means algoritmus véletlen kezdőpontból indul, tehát többszöri elvégzésre más-más eredmények születhetnek. A dolgozatomban mindig az elsőként létrejött



5.13. ábra. A k-means, k-SPLITOUR, k-SVPP alkalmazása nyolc robotra



5.14. ábra. A k-means, k-SPLITOUR, k-SVPP alkalmazása kilenc robotra



5.15. ábra. A k-means, k-SPLITOUR, k-SVPP alkalmazása tíz robotra

képet mutatom be, de lehetséges lehet ennél, jobb és rosszabb klaszterezés az egyes robotszámokra. A $k \in [6, 10]$ robotra való szimulációt tekintve arra lehetünk figyelmesek, hogy a nagyobb k értékek mellett az átlagos eltérés kisebb, azonban az egyes robotok által megtett úthosszak, és azok maximuma is csak kismértékben csökken, ugyanakkor a robotok által megtett összútvonal rohamosan nő. A k növekedésével látható, hogy a talált klaszterek már egyre kevésbé rendezettek, közelebb, távolabb helyezkednek el a bázis csomóponttól,

k										
1	2275,78									
2	1334,00	1097,80								
3	953,45	744,63	875,69							
4	575,33	678,19	667,53	732,52						
5	431,17	513,86	575,04	652,90	675,97					
6	542,70	527,73	425,75	346,22	627,29	549,00				
7	351,65	475,26	536,68	502,59	431,53	441,99	425,88			
8	419,21	489,26	425,88	371,90	518,62	378,30	383,96	449,73		
9	487,46	320,29	473,25	347,15	422,60	331,94	373,19	396,41	465,36	
10	321,08	432,08	406,04	347,15	325,35	371,61	465,36	383,07	376,43	449,29

5.1. táblázat. A k-means algoritmusban kapott alútvonalak hossza [m]- ben megadva

k										
1	2275,78									
2	1245,00	1204,10								
3	850,39	820,17	886,79							
4	732,98	675,02	674,97	771,06						
5	633,26	601,51	562,07	565,73	614,23					
6	610,35	468,40	539,41	466,87	501,10	549,61				
7	492,46	464,72	465,01	416,12	418,07	423,41	524,48			
8	492,46	412,24	428,00	452,57	336,00	418,19	436,13	475,18		
9	439,77	422,60	342,21	466,43	416,12	289,77	429,09	344,35	475,18	
10	439,77	375,80	345,95	372,57	441,25	320,89	320,36	438,66	362,49	429,27

5.2. táblázat. A k-SPLITOUR algoritmusban kapott alútvonalak hossza [m]- ben megadva

k										
1	2275,78									
2	1275,00	1202,10								
3	832,98	823,21	891,23							
4	731,95	689,93	583,44	777,94						
5	607,61	561,14	547,39	557,39	618,35					
6	577,56	446,65	539,41	441,40	503,71	537,10				
7	481,13	485,60	486,87	402,35	414,90	423,41	514,10			
8	481,13	412,24	398,64	452,33	326,07	418,19	463,88	474,54		
9	437,55	422,60	329,99	453,98	416,12	298,12	429,09	359,23	476,21	
10	437,55	375,80	337,30	372,57	448,21	297,75	320,36	450,73	371,71	428,80

5.3. táblázat. A k-SVPP algoritmusban kapott alútvonalak hossza [m]-ben megadva

gyakori, hogy az egyik távolabbi klaszterhez tartozó útvonal a közelebbieket több helyen is metszi. A 5.14 és 5.15 ábrákon látható, hogy az utóbbin sárgával jelölt 1-7-35-23-21-36 csomópontokat tartalmazó klaszter és a lilával jelölt 1-6-34 csoport a két ábrán azonos. Tehát nagy k-ra előfordulhat, hogy az egyes klaszterek ugyanúgy alakulnak ki különböző számú robotot alkalmazva.

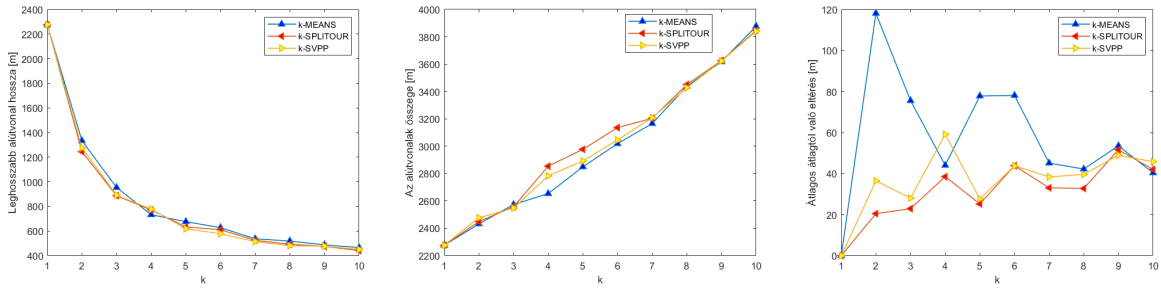
A k-SPLITOUR és a k-SVPP algoritmusokban, gyakori, hogy a nem egy klaszterben elhelyezkedő csomópontok közel helyezkednek el egymáshoz, az útvonalak sok helyen metszik egymást, ami a későbbiekben, a robotok időbeli mozgásának megtervezését tekintve extra nehézségeket jelenthet. Ezzel szemben a k-means, főleg viszonylag kevés számú robot esetén elkülönülő klasztereket képez, amelyeken megtervezett útvonalak ritkán keresztezik egymást. A k-SPLITOUR már két robot használata esetén is keresztező útvonalat hoz lét-

re 5.7 ábra, a k-SVPP három robotra 5.8 ábra, míg a k-means csak négy robotra 5.9 ábra hoz létre kereszteződő útvonalakat. Itt a bázis csomópontnál levő kereszteződéseket nem vettem figyelembe, hiszen egy bázis csomópont használata esetén ezek elkerülhetetlenek.

A k-SPLITOUR vágási pontjainak követése könnyen megtehető az ábrákat tekintve (5.7 - 5.15 ábra középső képek). Az egy robotos megvalósítást tekintve a robot a bázisállomásból a 11. érzékelő csomópontba halad, tehát ebbe az irányban járja be a pályát. A két robotos bejárás során az első a 32. csomópontról gyűjt utoljára adatot az 5.7 ábrán látható módon. Ekkor a két útvonal közti különbség körül-belül $40\,m$, amely $10s$ időtartam különbségnek felel meg. Az 5.7 – 5.9 ábrák középső képét tekintve látható, hogy $k = 2$ esetén a 27-19-18-9-10-32 csomópontok még a $Cluster_1$-hez, $k = 3$ esetén már $Cluster_2$-höz tartoznak. Ugyanígy $k = 3$ esetén a 12-31-37 érzékelő csomópontok még $Cluster_1$-hez, $k = 4$ esetben már a $Cluster_2$-höz tartoznak, és ugyanígy követhető a többi klaszterre és csomópontra is az algoritmus működése.

A k-SPLITOUR algoritmusnál gyakori, hogy az alútvonalakban hurok alakul ki. Ez azért fordul elő, mert az egy robotra meghatározott utat csak szétvágjuk, a csomópontok sorrendjét nem változtatjuk. Sok robotot alkalmazva észrevehető, hogy a robotszámot eggyel növelve nem változik meg minden útvonal. Ugyanis ilyenkor, esetlegesen az akadályok elhelyezkedése, vagy a csomópontok távolsága miatt a vágási feltétel nem teljesülne akkor, ha a következő csomópont is tagja lenne az adott klaszternek. A k-means algoritmussal ellentétben, a k-SPLITOUR a klaszterek meghatározásakor figyelembe vesz akadályokat, de csak az egy robotos útvonal mentén. Tehát a vágási pontok szélén elhelyezkedő, és a bázis csomópont közötti akadályokat ez sem veszi számításba. De mivel az akadályok ilyen módon figyelemebe veszi, és a klaszterek meghatározása után már nem változtat az útvonalon az alútvonalak hosszának átlagos eltérése ennél az algoritmusnál a legkisebb, tehát a pálya bejárása során ezt alkalmazva lesz a csomópontok késleltetési ideje közel ugyanannyi a különböző robotok által bejárt csoportokra.

A k-SVPP algoritmus a k-SPLITOUR algoritmus klaszterezését alkalmazva tervezi meg M-SVPP algoritmusok alkalmazásával az egyes alútvonalakhoz tartozó lehető legrövidebb útvonalat. Ez legtöbbször az egyes hurkok kisimitását jelenti mint például az 5.11 ábra 2-3 képén a lila és a középkék útvonalak esetén. A táblázat adatai alapján ez a középkék (1. alútvonal) esetén körül-belül $610\,m - 578\,m = 32m$, míg a lila (4. útvonal) esetén $467\,m - 441\,m = 26\,m$ úthossz spórolást jelent. Ugyanezen az ábrán látható, hogy a sárga, a világoskék és a bordó útvonalak esetén a k-SVPP nem készített új csomópont bejárási sorrendet az alútvonal bejárásához, tehát itt a k-SPLITOUR algoritmusban meghatározott csomópont permutációk kerülnek alkalmazásra.



5.16. ábra. A k-means, k-SPLITOUR, k-SVPP, jellemzése $k \in [1, 10]$ robotot alkalmazva: Az alútvonalak közül a maximális hossza, az bejárási úthosszak összege, és az átlagos eltérés.

Az 5.16 ábra első képén az egyes alútvonalak közül a leghosszabb került ábrázolásra. Ezzel az úthosszal arányos ugyanis a teljes bejárási idő, tehát az az időtartam amely alatt az összes érzékelő csomópontról összegyűjtik és feltöltik az adatot a bázis csomópontra a robotok. Az ábra alapján, a vártnak megfelelően minél több robot gyűjt adatot annál rövidebb lesz ez a bejárási idő. Azonban egyre több robotot alkalmazva a bejárási idő csökkenése egyre kisebb mértékű, ugyanis a bázis csomóponthoz mindig vissza kell térnie az egyes robotoknak ami egyes esetekben jelentős időt vehet igénybe az adatgyűjtés folyamán. A legrövidebb bejárási időt a k-SVPP biztosítja azonban ettől csak kis mértékben tér el a k-SPLITOUR alkalmazása. Ez nyilván valóan attól függ, hogy milyen a terep elrendezése, és az egyes vágások után milyen a keletkezett bejárási sorrend. Ha a vágások után csak kis mértékben, vagy nem kerülnek átrendezésre a k-SVPP alkalmazása során az egyes alútvonalakban a csomópontok sorrendje akkor közel azonos útvonalak keletketnek a k-SPLITOUR és a k-SVPP során. Azonban az is előfordulhat, hogy a k-SVPP hosszabb bejárási időt biztosít mint a k-SPLITOUR az akadályok elhelyezkedése miatt. Például 9-10 robot esetén az ábrán. Ugyanis a k-SPLITOUR algoritmus a P egy robotra tervezett út mentén figyelembe veszi az akadályok elhelyezkedését, azonban a k-SVPP a csomópontok átrendezése, TSP megoldása során csak a csomópontokat veszi figyelembe, az akadályoktól akkor eltekint, és csak a későbbi lépésekbe veszi be őket az útvonal tervezésebe. A 5.14 és 5.15 ábrákon a 9 és 10 robotot alkalmazó esetek kerültek ábrázolásra. Észrevehető, hogy ilyenkor a k-SVPP és a k-SPLITOUR csomópont permutációi közel azonosak. A 10 robotot alkalmazó esetnél a világoskék (5. alútvonal) hosszabb a k-SVPP algoritmusnál a 2. és 6. akadályok elhelyezkedése miatt. Ugyanezen az ábrán a rózsaszínű (9. alútvonal) alútvonalaknak ránézésre azonos a bejárási sorrendje a k-SPLITOUR és k-SVPP algoritmusokban, azonban az útvonal hossza a k-SVPP algoritmus esetén 12 m-rel hosszabb. Ennek az a magyarázata, hogy a k-SVPP algoritmus a TSP megoldása során megfordította a csomópontok sorrendjét, és az előre meghatározott negatív kezdőirány következtében a k-SVPP algoritmus során a robot kénytelen kevésbé optimális útvonalon haladni.

Az 5.16 ábra második képén az alútvonalak összege látható, tehát az egyes robotok útvonalainak összege. Ezen látható, hogy ugyan a bejárási idő folyamatosan csökken, a robotok által összesen megtett út folyamatosan nő. Észrevehető, hogy a 7-8-9-10 robotra tervezett útvonal bejárási ideje alig csökken, az összesen bejárandó úthossza viszont egyenletesen nő. Ezért az alkalmazandó robotok száma ezek alapján megfontolandó, figyelembe véve a rendelkezésre álló erőforrásokat, és célokat. A 2-3 és 7-8-9-10 robotra a három algoritmus közel azonos útvonalösszeget nyújt. Azonban 4-5-6 robotra a k-SPLITOUR összútvonala a legnagyobb, míg a k-means algoritmusal számolté a legkisebb. A k-means algoritmusnál ugyanis a csomópontok klaszteren belüli távolságát csökkentjük, tehát olyan klasztereket határozunk meg, ahol a csomópontok egymáshoz a lehető legközelebb helyezkednek el. A k-SVPP algoritmus pedig a k-SPLITOUR algoritmusban megkapott klaszterekben keresi a lehető legrövidebb útvonalat tehár ezek összege várhatóan kisebb mint a k-SPLITOUR által meghatározott. 7-8-9-10 robotot alkalmazva az összútvonalbeli különbség azért csökken le, mert a k-means algoritmusban létrejött klaszterek közelsem egyenlő távolságra helyezkednek a báziscsomóponttól, a k-SVPP algoritmus pedig az akadályok elhelyezkedése és a bázis csomóponthoz való visszatérés miatt nem, vagy nem jelentős mértékben tud rövidebb alútvonalakat létrehozni.

Az 5.16 ábra harmadik képe az egyes alútvonalak átlagos eltérését mutatja. Ez arra nyújt betekintést, hogy az egyes robotok bejárási ideje mennyire különbözik egymástól. Ha nagy az átlagos eltérés, akkor nagy a bejárási úthossz különbség, tehát az egyes csomópontokról az adatok beérkezésének a késleltetési ideje különböző attól függően, hogy mely klaszterben helyezkednek el. A k-SPLITOUR algoritmusnál legkisebb az átlagos eltérés, hiszen ezen algoritmusnál az a cél, hogy közel azonos hosszú alútvonalakra bontsa fel a megoldást. A k-SVPP algoritmusnál a k-SPLITOUR-rel létrehozott csomópontok

bejárási sorrendjét átalakítjuk ezért az egyes klaszterekben különböző lesz a bejárási útvonal hossza, ezért nagyobb az átlagos eltérés. A k-means algoritmusnál az átlagos eltérés nagymértékű, hiszen a k-means algoritmus arra optimalizál, hogy a klaszterekbe tartozó csomópontok egymáshoz való távolsága legyen minimális, nem pedig arra, hogy a klaszterekben szereplő csomópontokra épített útvonalak, közel azonos hosszúak legyenek.

Összességében kijelenthető, hogy mindhárom, több robotra tervezett algoritmus jó eredményt ad, alkalmazásuk függ a bejárandó tereptől és a feladat céljától. Az alkalmazandó robotok száma szintén a megoldandó feladat céljától és az erőforrás rendelkezésre állástól függ.

6. fejezet

Összegzés

A dolgozatomban egy kerekű adatgyűjtő robotok számára terveztem útvonalat érzékelő csomópontok között. A feladat az összes érzékelő csomóponton összegyűjtött adat feltöltése a bázis csomópontra. A minél hatékonyabb adatgyűjtés érdekében a cél a bejárandó útvonal és a bejárási idő csökkentése, miközben biztosítani kell az ütközés mentességet mind a csomópontokhoz mind az akadályokhoz.

A megoldást először elkészítettem egy robot számára, ahol új algoritmusokat készítettem az akadályok kezelésére, illetve új feltételeket támasztottam az ütközésmentességre. Szintén új algoritmusokat állítottam elő a fagráf létrehozására, amelyben végső soron a lehető legrövidebb út keresése kerül végrehajtásra.

Ezt követően három algoritmust mutattam be a több robotot alkalmazó megoldásra. k-means klaszterező algoritmuson alapuló megoldást vezettem be. Ezek implementálása során az egy robotra bevezetett új algoritmusokat, feltételezéseket szintén alkalmaztam.

A dolgozatomat szimulációs eredmények részletes bemutatásával zártam. Először a terep előkészítését és a pályatervezés algoritmus lépéseit vázoltam fel ábrák segítségével. Majd elemeztem az egy robotot használó útvonal tervezés során kapott eredményeket. A munkám során összehasonlítottam a szakirodalomban eddig elért és az általam elkészített algoritmus eredményeit.

Az útvonal tervezés eredményeit bemutattam több robotot használó esetre is. Az itt elért eredményeket mind numerikusan mind vizuálisan ismertettem. A három különböző algoritmust összehasonlítottam, mérőszámokat definiáltam, amelyek alapján több szempontból vizsgálhatóvá váltak az algoritmusok. Ilyenek a pálya egészének bejárásához szükséges idő, a robotok által megtett úthosszak összege, illetve az átlagos eltérése az egyes robotok által megtett útvonalaknak, amely az adatok késleltetési idejének eltérésére mutat rá. Illetve felvázoltam, hogy az egyes algoritmusok, így mérőszámaik hogyan változnak az alkalmazott robotok számának függvényében.

A több robotot alkalmazó megoldás azonban még nem teljes, a feltüntetett eredménynek kismértékben manipulálhatóak a kezdőirányok megválasztásával. A közel egyforma hosszú minimális hosszúságú útvonalak létrehozásához a későbbiekben optimalizáló algoritmusok használatát javaslom.

Az akadályok kezelésére készített algoritmusom habár hatékonyabb mint az eddig ismert, azonban még mindig vannak hiányosságai, amit a szimulációk elemzésénél meg is mutattam részletesen.

Összefoglalva, az akadályok kezelésére általam elkészített algoritmus rövidebb útvonal elkészítésére alkalmas. A bemutatott három, több robotot alkalmazó algoritmusok mindegyike megfelelő a kitűzött feladat elvégzésére.

Köszönetnyilvánítás

Szeretném megköszönni a konzulensemnek, Dr. Harmati István Tanár Úrnak a rengeteg segítséget, a gyors és pontos válaszokat a kérdéseimre, a fantasztikus ötleteit és a rendszeres konzultációkat, amely nélkül ez a munka nem készülhetett volna el.

Irodalomjegyzék

[1] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, J. Anderson, Wireless sen- sor networks for habitat monitoring, in: The 1st ACM International Workshop on Wireless Sensor Networks and Applications, 2002, pp. 88–97

[2] T. He, S. Krishnamurthy, J.A. Stankovic, T. Abdelzaher, L. Luo, R. Stoleru, T. Yan, L. Gu, J. Hui, B. Krogh, Energy-efficient surveillance system using wireless sen- sor networks, in: the 2nd International Conference on Mobile Systems, Appli- cations, and Services, ACM, 2004, pp. 270–283

[3] I. Chatzigiannakis, A. Kinalis, S. Nikoletseas, Sink mobility protocols for data collection in wireless sensor networks, in: the 4th ACM International Work- shop on Mobility Management and Wireless Access, ACM, 2006, pp. 52–59

[4] Y. Yun, Y. Xia, Maximizing the lifetime of wireless sensor networks with mo- bile sink in delay-tolerant applications, IEEE Trans. Mob. Comput. 9 (9) (2010) 1308–1318

[5] Y. Gu, F. Ren, Y. Ji, J. Li, The evolution of sink mobility management in wireless sensor networks: a survey, IEEE Commun. Surv. Tut. 18 (1) (2015)

[6] Hailong Huang, Andrey V. Savkin, Viable path planning for data collection robots in a sensing field with obstacles, Computer Communications 111 (2017) 84–96

[7] Dubins, L. E. (1957). On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents. American Journal of Mathematics, 79(3), 497–516.

[8] Bellman, R. "On a Routing Problem." Quarterly of Applied Mathematics. Vol. 16, Number 1, pp. 87–90.

[9] Dijkstra, E. W. "A Note on Two Problems in Connexion with Graphs." Numerische Mathematik. Vol. 1, Number 1, 1959, pp. 269–271.

[10] G.N. Frederickson, M.S. Hecht, C.E. Kim, Approximation algorithms for some routing problems, in: SIAM Journal on Computing, 7, 1978, pp. 178–193

[11] S. Lloyd, "Least squares quantization in PCM," in IEEE Transactions on Information Theory, vol. 28, no. 2, pp. 129-137, March 1982

[12] Steven M. LaValle, Planning Algorithms: Dubins Curves (2006), http://planning.cs.uiuc.edu/node821.html, Cambrige University Press,2021.10.07

[13] Brazil, Marcus and Grossman, Peter and Rubinstein, Hyam and Thomas, Doreen, Demonstrating efficiency gains from installing truck turntables at crushers, ResearchGate,2015

[14] J.T. Isaacs, J.P. Hespanha, Dubins traveling salesman problem with neighbor- hoods: a graph-based approach, Algorithms 6 (1) (2013) 84–99

[15] A.M. Frieze, G. Galbiati, F. Maffioli, On the worst-case performance of some algorithms for the asymmetric traveling salesman problem, Networks 12 (1) (1982) 23–39

[16] Diaby, Moustapha, The traveling salesman problem: A Linear programming formulation of, CoRR, 2006

[17] Traveling Salesman Problem: Solver-Based: https://ch.mathworks.com/help/ optim/ ug/ travelling-salesman-problem.html 2021.05.08.